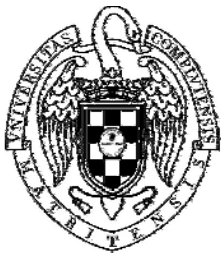


Sistemas Informáticos
Curso 2002-03

Televisión digital

Rodrigo Borrego Alonso
Sara Hernández García
José Luis Moisés González
Carolina Valdazo Ampuero

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

Facultad de Informática
Universidad Complutense de Madrid

Extracto

El objetivo de este proyecto es el análisis, diseño e implementación sobre una FPGA de altas prestaciones (Xilinx Virtex XSV-800) de un sistema de procesado digital en tiempo real de una señal de vídeo digitalizada previamente, y su visualización sobre un monitor VGA estándar. El procesado incluye la aplicación de efectos sobre la imagen tales como pausa, zoom, mosaico, rotación, reflejo, estroboscópico y otros. La implementación se da en lenguaje vhdl.

Este documento introduce primero los fundamentos tecnológicos de la imagen digital y del tipo de hardware empleado, así como diversos estándares sobre los que se basan determinadas decisiones de diseño. Tras una breve presentación del escenario general de un sistema digital de procesamiento de imagen, se detallan el diseño y descomposición modular de nuestro sistema particular, donde todo el procesamiento se realiza sobre formato RGB. Las conclusiones valoran las principales fases del proyecto, junto con las ventajas e inconvenientes encontrados en la plataforma de desarrollo elegida.

Todo el código vhdl se acompaña en un apéndice.

Abstract

This project aims to analyse and design a digital image-processing system to be implemented on a high performance FPGA. This is a real time system which takes a digitalised video signal as input to produce a VGA output onto a standard monitor.

Several effects on the image such as still, zoom, mosaic, rotation, flip, strobe, and some other, are available. Vhdl is the language of implementation.

First, this document introduces the technical background of digital image as well as that of system-specific hardware, along with a short overview of different video standards that posed some design decisions. After a brief introduction to the scenario of digital image-processing systems, there follows a detailed description of our design and its modular decomposition. Within the project, all processing is done using RBG format. Conclusions deal not only with the main project milestones but also with the advantages and disadvantages that came out of our specific development platform.

An appendix comprises the complete vhdl source code.

Índice

1. Introducción

La tecnología analógica ha venido utilizándose tradicionalmente en la práctica totalidad de las áreas de ciencia y conocimiento. De un tiempo a esta parte, está siendo sustituida paulatinamente por la tecnología digital, que cada vez abarca mayor campo de aplicación. En particular, este cambio cobra especial relevancia en el tratamiento de imágenes, ya que afecta a todo el proceso: adquisición, procesado y reproducción.

Entre otras, hay dos razones principales que han motivado esta tendencia: el aumento de capacidad de las tecnologías digitales, y la aparición de nuevas técnicas y algoritmos. Las nuevas capacidades: mayor velocidad de procesamiento, mayor ancho de banda en las comunicaciones, y mayores posibilidades de almacenamiento, hacen posible la aplicación de nuevos algoritmos, más optimizados que requieren menos recursos.

De este modo, se contribuye y favorece doblemente el desarrollo del campo digital que poco a poco va implantándose con mayor fuerza y ocupando más áreas dentro del tratamiento de las imágenes, imponiéndose progresivamente al tratamiento mediante técnicas analógicas.

El objetivo de este proyecto es, pues, el análisis, diseño e implementación sobre una FPGA de altas prestaciones (Xilinx XSV-800) de un circuito específico para el procesado digital en tiempo real de una señal de video previamente digitalizada y su visualización sobre un monitor VGA estándar.

Dentro del procesado de la imagen se incluyen, entre otros efectos: la pausa y memorización, efecto estroboscópico, visualización simultánea de varias imágenes, efectos zoom y mosaico, rotación e imagen especular, etc.

Las prestaciones de la placa son muy altas y dispone de un gran número de posibilidades (USB, Ethernet, PS/2, VGA, varios bancos de memoria...), sin embargo, al tratarse de una placa de prototipado de propósito general, en determinados momentos ha condicionado el diseño e implementación de nuestro circuito debido a algunas limitaciones inherentes a esa generalidad.

Por otro lado, durante el desarrollo del proyecto hemos tratado de adaptarnos lo máximo posible a los estándares actuales intentando que el circuito obtenido sea lo más *real* posible.

El contenido de esta memoria lo podemos dividir en tres partes claramente diferenciadas. La primera, introduce las tecnologías implicadas en todo el proceso de desarrollo de este proyecto (FPGA, imágenes, monitores, ...). La segunda, analiza y detalla el diseño e implementación obtenidos, haciendo hincapié en los aspectos más relevantes y justificando las decisiones adoptadas. La tercera, resume las conclusiones e indica posibles desarrollos posteriores. El código fuente que constituye el prototipo se adjunta a modo de apéndice.

2. Aproximación Tecnológica

2.1. Fundamentos de la Imagen Digital

Al igual que con otros muchos elementos, las imágenes las podemos considerar desde la perspectiva analógica o la digital. Las **imágenes analógicas** se crean a partir de dos factores: brillo (luminosidad) y color (que tiene tres subfactores: rojo, verde, azul). Las **imágenes digitales**, por el contrario, no necesitan toda esta información, porque se crean a partir de unos datos básicos y el resto de calcula a partir de éstos. Esto hace que cada vez se tienda más a elementos digitales en cada uno de los procesos relacionados con el tratamiento de las imágenes.

El proceso de **digitalización** consiste en dividir la imagen continua a(x, y) en N filas y M columnas, donde cada uno de los M x N elementos generados se denomina píxel. El valor asignado a cada uno de esos píxeles puede ser considerado como la evaluación de la señal física que choca contra el sensor de dos dimensiones que captura la imagen, la cual es una función de diferentes variables, incluyendo la profundidad, el color y el tiempo.

Hay diferentes valores para los parámetros M y N que usualmente vienen determinados por los estándares de video, por los requerimientos de los algoritmos o por el deseo de hacer los circuitos digitales un poco más simples. Los valores comúnmente encontrados para el número de filas (N) son: 256, 512, 525, 625, 1024 y 1035, y para el número de columnas (M): 256, 512, 768, 1024 y 1320.

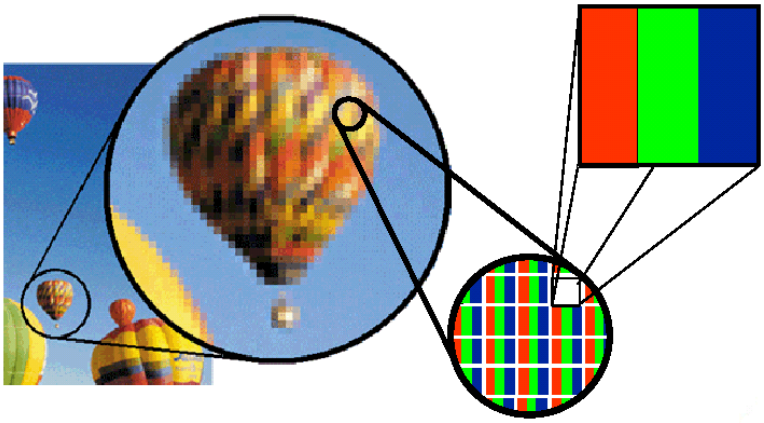
Cada **imagen** individual se considera como la suma aditiva de colores siendo los tres básicos cuando hablamos de televisión, el rojo (RED), el verde (GREEN) y el azul (BLUE). Por eso los dispositivos que tratan con este tipo de imágenes se denominan **dispositivos RGB**.

Como ya hemos visto, tenemos imágenes que se consiguen a través de dispositivos analógicos (cine, cámaras de fotos, televisión, ...) y otras, a través de técnicas digitales (ordenadores personales, escáneres, ...).

Nosotros, debido al enfoque de nuestro proyecto, vamos a centrarnos en las imágenes digitales y su tratamiento.

2.1.1. Anatomía de una Imagen Digital

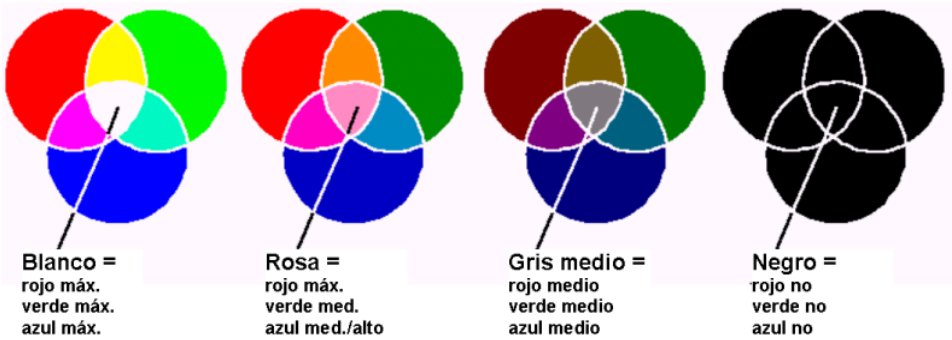
En el entorno digital, las imágenes están formadas por elementos mínimos con unas determinadas características físicas: colores (rojo, verde y azul) y brillo. Cada uno de estos puntos de la imagen se llama **píxel** y cuanto mayor sea el número de ellos que haya en un determinado área de la imagen, mayor será la **resolución**.



Para cada uno de estos píxeles, tenemos que almacenar cierta información que nos permita recuperarlo después. Lo más habitual es aprovechar la descomposición de los colores para almacenar la información completa del píxel teniendo en cuenta el valor para cada una de las **componentes de color**. En función del número de bits que le asignemos a cada una de estas componentes (generalmente es el mismo para las tres), mayor número de colores podremos representar. Al conjunto de colores representables, se le suele denominar **paleta de colores**.

Generalmente, se habla de "**color real**" cuando tenemos una paleta de 16'8 millones de colores, para lo que necesitamos manejar 24/32 bits para cada uno de los píxeles de nuestra imagen (8 bits para cada una de las componentes). Sin embargo, dada la enorme cantidad de información que esto requiere, se suelen utilizar otras aproximaciones, que se centran básicamente en la reducción del número de bits que le dedicamos a cada una de las componentes y entre las que hay que destacar la que asigna 5:5:5 (ó 5:6:5) bits para cada una de ellas (en total 15/16 bits) y que nos permite alcanzar paletas de 65 mil colores.

Merece una mención especial una aproximación que reduce enormemente el número de información a manejar (únicamente requiere 8 bits por píxel, por lo que tiene una paleta de 256 colores) a costa de añadir una serie de cálculos adicionales y que ha recibido el nombre de **pseudo color**.



El **tamaño** requerido por una imagen (en bits) estará directamente relacionado con el numero de píxeles que haya (resolución) y por la cantidad de información que necesite cada uno:

Tamaño de la Imagen = Número total de píxeles x Número de bits por píxel

Por lo tanto, a la hora de decidir qué formatos vamos a elegir (resolución, paleta de colores, ...), tendremos que tener en cuenta el sistema del que disponemos, para ver si va a ser capaz de manejar las cantidades de información requeridas, ya no sólo espacialmente, sino temporalmente.

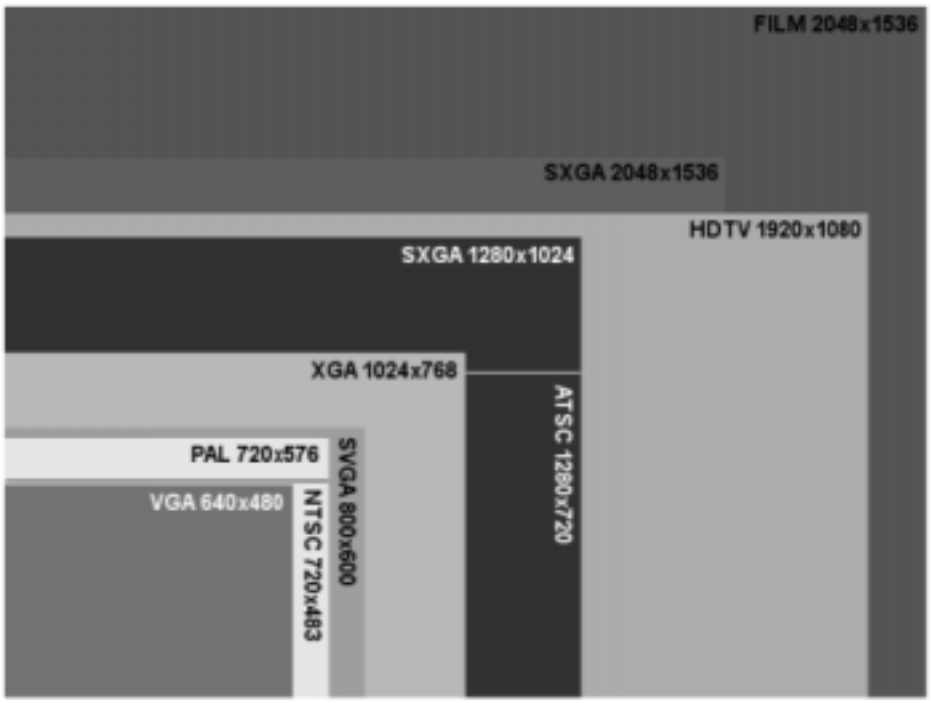
2.1.2. Resolución de una Imagen Digital

La **resolución en los sistemas de televisión** se especifica en términos del *número de líneas*. Este parámetro se determina observando un *test* de un patrón que consiste en alternar líneas blancas y negras que se ponen muy cerca entre sí; el par de líneas con el espaciado más cercano que pueda distinguirse como líneas separadas determina la resolución. Las líneas que pueden ser extrapoladas de un lado a otro de la pantalla con un ancho igual a la altura de una imagen son las *líneas de resolución*.

La **resolución en los formatos de los ordenadores** se especifican normalmente en función del número visible de píxeles en las dimensiones horizontal y vertical. Por ejemplo, una señal en formato VGA tiene 640 píxeles visibles en la dirección horizontal y 480 píxeles visibles en la dirección vertical, mientras que una señal en formato XGA tiene 1024 píxeles visibles en la dirección horizontal y 768 píxeles visibles en la dirección vertical.

A la hora de hablar de estas resoluciones, solemos nombrarlas dando su número de píxeles en horizontal y en vertical (o con el nombre del estándar que representa). Así, algunas de las más comunes son: 648x480 (VGA), 800x600 (SVGA), 1280x1024 (SXGA), etc. Haciéndolo así, conseguimos abstraer la idea de resolución y hacerla independiente del dispositivo que estemos usando para representar la imagen, pues únicamente hacemos referencia al número de píxeles que se usará en cada una de las dos dimensiones.

Las más representativas se pueden ver en la siguiente figura donde además, se puede ver la relación entre sus tamaños.



2.1.3. Formatos de Imagen Digital

Hay muchos tipos diferentes de señales de video, pero las podemos dividir en dos grandes grupos: de televisión y de computación.

Los formatos de las señales de televisión varían de país a país y se distinguen tres subgrupos:

- En los Estados Unidos y Japón se usa el formato **NTSC**, que significa *National Television Systems Committee*, que es el nombre de la organización que desarrolló el estándar.
- El formato más común en Europa es el **PAL** (*Phase Alternating Line*), que se desarrolló con base en el NTSC, y contiene algunas mejoras sobre este último.
- El formato **SECAM** se usa en Francia, y significa *SEquential Coleur Avec Memoire* (con memoria).

Existen cerca de 15 subformatos contenidos en estos tres generales, donde cada uno, normalmente es incompatible con los demás.

Aunque todos ellos utilizan el mismo sistema de barrido y representan el color con un tipo de modulación en fase, difieren específicamente en las frecuencias de barrido, el número de líneas y la técnica de modulación del color, entre otras.

Los diversos **sistemas de computación** como VGA, XGA o UXGA también difieren sustancialmente, donde la principal diferencia son las frecuencias de barrido.

Estas diferencias no causan mucha preocupación, ya que la mayoría de los equipos de computación están siendo diseñados para soportar diferentes frecuencias de barrido.

En la siguiente tabla podemos ver las frecuencias y resoluciones típicas para los principales formatos de televisión y computación.

FORMATO DE VIDEO	NTSC	PAL	HDTV/SDTV	VGA	XGA
Descripción	Formato de televisión para Norte América, Centro América, parte de sur América y Japón	Formato de televisión para la mayoría de Europa y Sur América	High Definition/ Standard Definition Digital Television Format	Video Graphics Array (PC)	Extended Graphics Array (PC)
Resolución Vertical (líneas visible por *frame*)	Aprox. 480 (525 líneas en total)	Aprox. 575 (625 líneas en total)	18 formatos diferentes: 1080 o 720 o 480	480	768
Resolución Horizontal (píxeles visibles por línea)	Determino por el ancho de banda, rangos desde 320 hasta 650	Determino por el ancho de banda, rangos desde 320 hasta 720	18 formatos diferentes: 1920 o 704 o 640	640	1024
Frecuencia Horizontal (KHz)	15.734	15.625	33.75-45	31.5	60
Frecuencia Vertical (Hz)	29.97	25	30-60	60-80	60-80
Frecuencia más elevada (MHz)	4.2	5.5	25	15.3	40.7

2.1.4. Interfaces de Video

Existen tres niveles básicos de señales de interfaz de banda base. En el orden de incremento de la calidad son: **compuesto** (o CVBS), que utiliza un par de cables; **Y/C**, que emplea dos pares de cable; y **por componentes**, que emplea tres pares de cables; donde cada par de cables consiste en una señal y una tierra.

Estas tres interfaces difieren en sus niveles de combinación de la información (codificación). Usualmente a mayor codificación se degrada la calidad, pero permite que la señal se transmita con menos cables y tenga menos requisitos. La interfaz por componentes tiene la menor codificación, mientras que la interfaz compuesta posee la mayor.

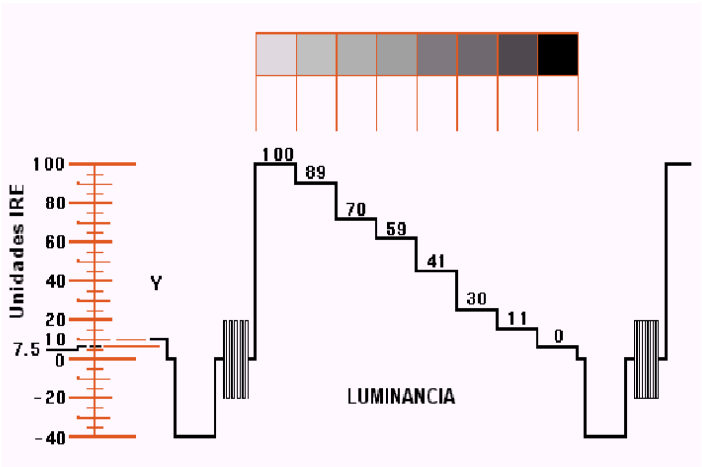
2.1.4.1. Interfaz Compuesta o CVBS

Las señales de esta interfaz son las usadas en el video analógico. El **video compuesto** también se llama **CVBS (*Color, Video, Blanking y Sync*)**, o señal banda base de video compuesto.

Esta interfaz combina la información de brillo (luminancia), la información de color (croma) y las señales de sincronización en un solo cable. El conector de estos cables es usualmente una toma RCA.

Cada línea esta formada por el *video activo* y el intervalo de *blanking*. El video activo es la parte de la señal que contiene la información del brillo (luminancia) y color (croma) de la imagen. La luminancia es la amplitud instantánea de cualquier punto en el tiempo. La amplitud se mide en términos de una unidad llamada IRE, que es una unidad arbitraria donde 140 IRE son iguales a un voltio pico a pico. El voltaje durante la parte del video activo produce una imagen con brillo totalmente blanco para esta línea, mientras que la parte del intervalo de *blanking* se mostraría como negro, por lo tanto no se vería en la pantalla.

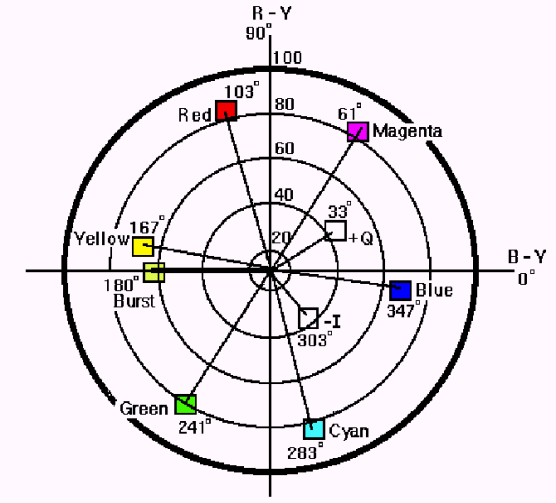
En la siguiente figura se muestra una señal de una línea de video que presenta una barra de diferentes luminosidades, y la relación de estas en unidades IRE.



La señal de luminancia es suficiente para obtener una imagen en escala de grises.

La información de color se añade en la parte superior de la señal de luminancia, y es una onda senoidal con los colores especificados por la diferencia de fase entre ésta y la fase de la señal de referencia llamada *burst*.

En la siguiente figura se muestran los principales colores y el *burst* con su correspondiente ángulo de fase.



La amplitud de la modulación es proporcional a la cantidad de color (o saturación), y la información de la fase indica el tono (o *hue*) del color.

La parte del *blanking* contiene el pulso de sincronización horizontal, así como la referencia del color *(burst*), que esta ubicado justo después del flanco de subida del pulso de sincronización horizontal.

Es importante notar que la parte de la señal correspondiente al intervalo de *blanking* horizontal está ubicada en el tiempo en el cual la señal no es visible, por lo que no se observa en la pantalla.

2.1.4.2. Interfaces Y/C

La señal Y/C es una señal de video un poco menos codificada. A menudo se llama incorrectamente *S-video*, pero este termino en la actualidad se emplea para referirse al formato de grabación de cintas para video, y no a una señal de interfaz.

El brillo (luminancia) que es la señal Y, y el color (croma), la señal C, se llevan en dos pares de cables diferentes.

El conector es usualmente un *mini DIN*, que parece una versión pequeña de un conector de teclado.

2.1.4.3. Interfaz por Componentes

Con estas interfaces se logra el más alto rendimiento, debido a que tienen la menor codificación, pues las señales están casi en su formato natural. Siempre emplean tres pares de cables que son típicamente cualquiera de estos dos: un formato que emplea una señal para la luminancia (Y) y dos señales diferenciales de color (R-Y, B-Y); o emplea una señal para el color rojo (R), otra para el verde (G) y otra para el azul (B), el cual comúnmente es llamado RGB.

Los **formatos RGB** casi siempre se emplean en aplicaciones de computación, mientras que los formatos diferenciales de color son generalmente usados en aplicaciones de televisión.

En los **formatos diferenciales de color**, la señal Y contiene la luminancia y la información de sincronización, y las señales diferenciales de color contienen el rojo (R) menos la señal Y, y el azul (B) menos la señal Y. La teoría detrás de estas combinaciones es que cada componente de color base (R, G o B) puede ser obtenida de estas señales diferenciales.

Existen algunas variaciones de estas interfaces diferenciales, como:

- **Y, B-Y, R-Y:** Señales de luminancia y diferenciales de color.
- **Y, Pr, Pb:** Pr y Pb son versiones escaladas de B-Y y R-Y. Comúnmente se encuentran en equipos de alto rendimiento.
- **Y, Cr, Cb:** Señal digital equivalente a Y, Pr, Pb. Algunas veces se usan incorrectamente en lugar de Y, Pr, Pb.
- **Y, U, V:** No es una interfaz estándar. Estas son intermediarias, entre las señales en cuadratura usadas en la formación de video compuesto y señales Y/C.

2.1.4.4. Interfaces para Ordenadores

Prácticamente todas las interfaces de ordenador emplean el formato RGB. La información de la imagen se transporta separadamente en las tres componentes bases de rojo, verde y azul. Además, la información de sincronización horizontal (H) y sincronización vertical (V) se lleva separadamente en dos señales.

Las cinco señales, R, G, B, H y V, se llevan en un cable que esta compuesto de un manojo de cables blindados; y el conector es casi siempre un tipo D de 15 pines. En algunas ocasiones, la información de sincronización horizontal y vertical se combina con una de las señales RGB, comúnmente la señal del verde, y a esto se le ha llamado *sync on green*. En raras ocasiones se encuentra la información de sincronización mezclada con las señales del rojo o el azul.

2.1.5. Relación entre las Diferentes Interfaces y los Colores RGB

Todos los formatos de señales de video comunes se crean por la combinación de las señales rojo (R), verde (G) y azul (B). Las señales RGB se generan en las cámaras de video o equipo de cine que crea las imágenes de video; si estas tres señales se almacenan como tres señales separadas se incrementan notablemente los requerimientos del sistema de almacenamiento, ya que cada señal requeriría un ancho de banda igual y muy elevado.

En lugar de esto es posible tomar ventaja del sistema de visión humana para reducir los requisitos del sistema de almacenamiento, lo cual se realiza transformando las señales RGB en nuevas señales de video que pueden ser de banda limitada con mínimas pérdidas de calidad en la imagen percibida.

La señal de luminancia (Y) en el estándar original del sistema de televisión NTSC se crea con la siguiente ecuación:

$$Y = 0.30\ R' + 0.59\ G' + 0.11\ B'$$

Los coeficientes seleccionados para la ecuación anterior están relacionados con la sensibilidad de los ojos a los colores RGB. Esta señal de luminancia tiene más ancho de banda que las señales originales RGB, ya que los pequeños detalles, representados por las altas frecuencias son retenidos en la parte monocromática de la imagen.

Los tubos de rayos catódicos usados en los televisores no producen la luz linealmente con las señales de video, por lo tanto, los valores RGB usados en la ecuación se modifican actualmente mediante la *corrección de curvatura inversa gamma*.

Las tres señales RGB deben ser regeneradas en el monitor para poder guiar el disparo de los colores rojo, verde y azul en el tubo de rayos catódicos (CRT) y para hacer esto posible se requieren en total tres ecuaciones, una por cada componente de color. Las dos nuevas ecuaciones son intermediarias de las que representan las señales diferenciales de color R'-Y y B'-Y. En el sistema de transmisión de televisión NTSC estas señales diferenciales de color se combinan en dos señales de color llamadas I y Q.

$$I = 0.74\ (R'-Y) - 0.27\ (B'-Y) = 0.60\ R' - 0.28\ G' - 0.32\ B'$$
$$Q = 0.48\ (R'-Y) + 0.41\ (B'-Y) = 0.21\ R' - 0.52\ G' + 0.31\ B'$$

O de forma matricial:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.30 & 0.59 & 0.11 \\ 0.60 & -0.28 & -0.32 \\ 0.21 & -0.52 & 0.31 \end{bmatrix} * \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}$$

Hay que tener en cuenta que, la adquisición de la señal de video por medio de cualquier dispositivo se realiza en RGB y su visualización en un CRT también, por lo que todas las transformaciones intermedias que hagamos para procesarla, deberán invertirse en el extremo opuesto.

2.1.6. Vídeo Digital

En realidad, el movimiento de las imágenes es una "**ilusión**". Lo que ocurre realmente es que las imágenes fijas pasan a una velocidad tal, que pensamos que se mueven. Este efecto tiene lugar porque las imágenes son retenidas por el ojo humano, que las mantiene durante un tiempo y de esta forma, cuando llega la segunda, todavía tenemos la anterior, por lo que realmente estamos percibiéndolas como si se estuviesen produciendo de manera continua.

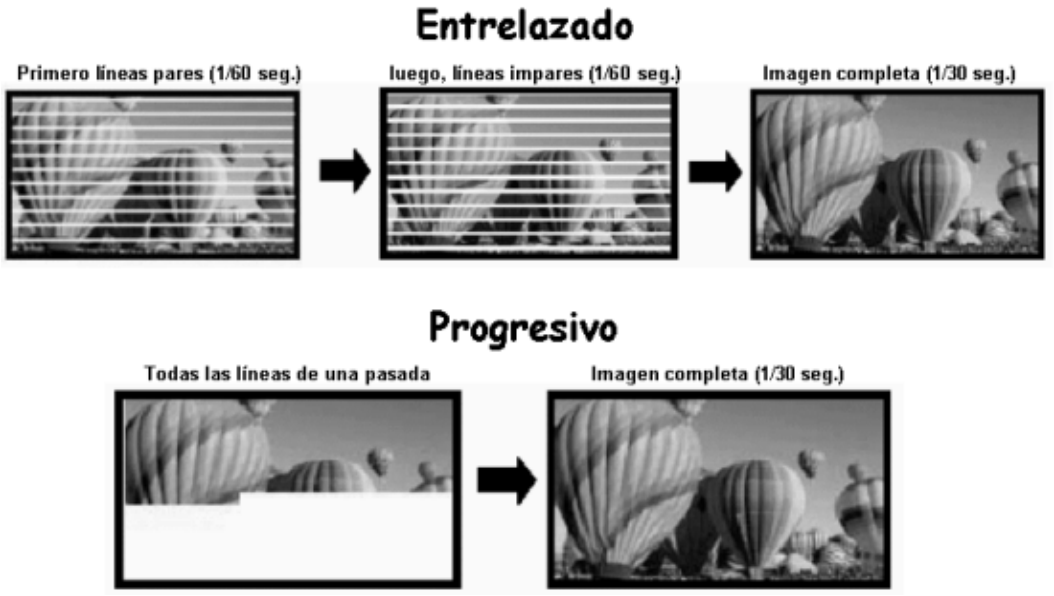
Para una buena calidad de imagen en movimiento, necesitamos una velocidad mínima de **24 imágenes por segundo**. Si reproducimos a menor velocidad, seremos capaces de percibir el parpadeo de las imágenes y, por encima, estaremos gastando inútilmente recursos, pues nuestro ojo no va a ser capaz de percibir más de 24. Por esto, la televisión analógica Europea emite 25 imágenes por segundo (en la Americana son 30 imágenes por segundo).

Una imagen se dibuja en un televisor o en una pantalla de un ordenador pasando una señal eléctrica horizontalmente a través de la pantalla (una línea a la vez). La amplitud de la señal en función del tiempo representa el brillo instantáneo en ese punto físico de la pantalla.

Hay dos formas de hacer estos barridos: entrelazado y progresivo.

En el **barrido entrelazado**, dividimos la imagen en dos campos, reproduciendo por separado y de manera consecutiva, primero las líneas impares y luego las pares, antes de pasar a la siguiente imagen.

Por el contrario, en el **barrido progresivo**, reproducimos línea a línea de manera secuencial la imagen de manera completa antes de pasar a la siguiente. Las diferencias entre ambos se ven claramente en la siguiente figura.



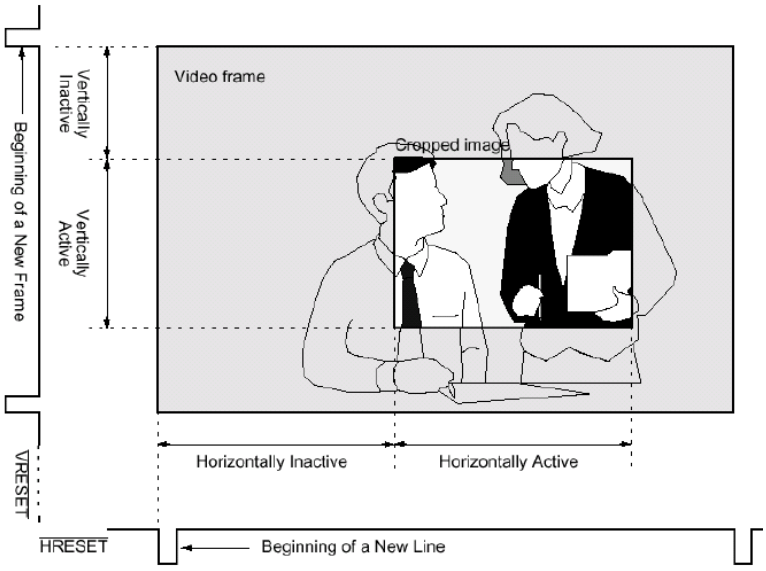
2.1.7. Monitores

Un **monitor** de ordenador es una **pantalla de tubo de rayos catódicos (CRT)**. Básicamente lo que hace es emitir varios haces de electrones que son redirigidos y que al chocar con el material fluorescente que lo recubre por dentro, hace que éste se ilumine de una u otra forma en función del tipo del material, la densidad del haz, etc.

Aunque hay otros métodos de generar imágenes sobre la pantalla, nosotros nos vamos a centrar en los **CRT de barrido**, en los que el haz barre de manera sistemática toda la superficie.

En un CRT de este tipo, el haz de electrones recorre la pantalla completa comenzando en la esquina superior izquierda y recorriendo progresivamente cada una de las líneas de izquierda a derecha. Cuando llega al final de una línea, apaga momentáneamente el cañón y se coloca al comienzo de la siguiente fila. Cuando todas las filas han sido recorridas y se ha llegado a la esquina inferior derecha, se apaga nuevamente el cañón y se regresa al punto de partida para comenzar con la siguiente imagen.

Todo este proceso se controla mediante tres señales: **sincronismo horizontal** (o de línea), que nos marca el inicio y final de cada una de las filas de píxeles; **sincronismo vertical** (o de frame), que nos indica el comienzo y el final de una imagen completa; e **intensidad**, que nos marca el valor a representar en el píxel que estamos refrescando. Para controlar el apagado del cañón, se suele usar una señal auxiliar (**blanking**) que hace que, independientemente del valor que se le esté dando, siga apagado, para evitar que durante esos períodos de retorno, el haz siga iluminando y aseguremos que está apagado.



2.1.8. Temporización VGA

Como acabamos de ver, nuestro ojo es capaz de percibir hasta 24 imágenes por segundo por lo que, para que seamos incapaces de percibir este parpadeo, tenemos que refrescar siempre por encima de estas velocidades.

Al igual que con las resoluciones y con las paletas de colores, este parámetro va a venir determinado por el sistema del que dispongamos y el estándar que queramos adoptar. Obviamente, aquí vuelve a jugar un papel muy importante el compromiso que tenemos que establecer entre los diferentes factores pues no tiene ningún sentido trabajar con frecuencias de refresco muy altas y resoluciones muy bajas o viceversa.

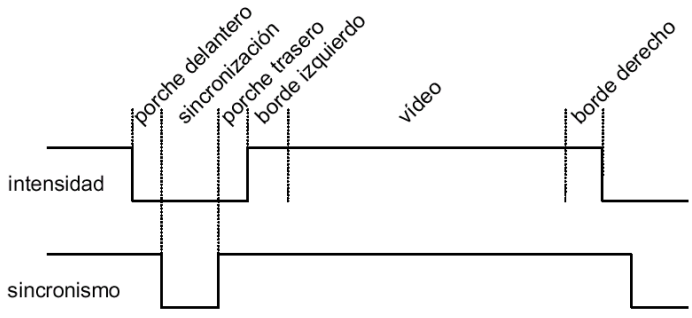
Estas frecuencias se suelen expresar en número de imágenes por segundo (o en Hertzios) y las más habituales son 50Hz, 60Hz y 75Hz. En nuestro caso, como vamos a adoptar el estándar VGA tenemos, como ya hemos visto, una resolución de 648x480 y una velocidad de refresco de 60 imágenes por segundo.

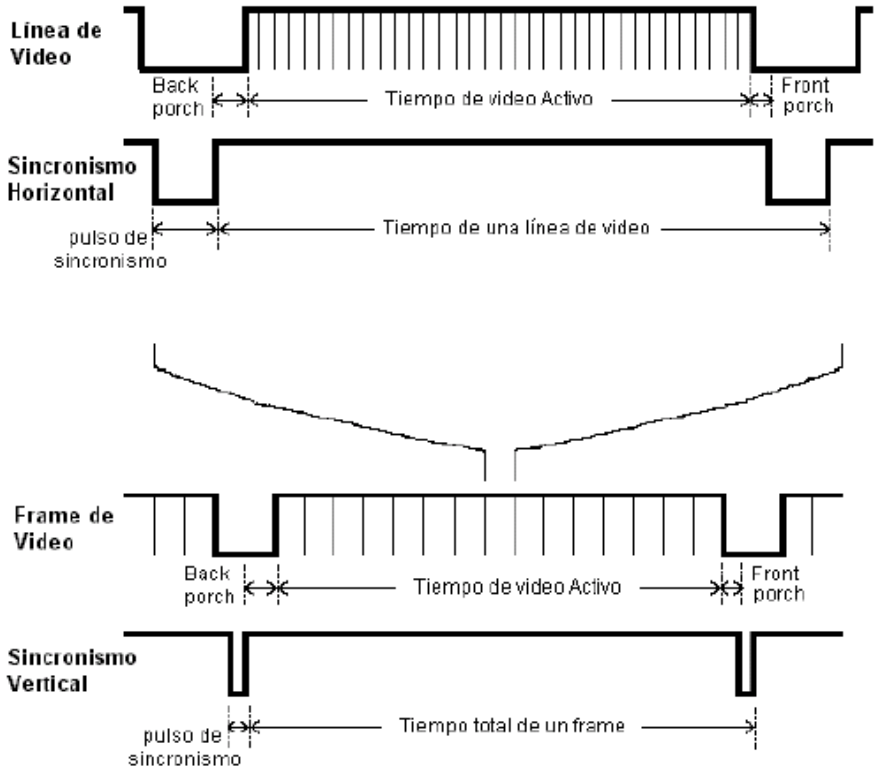
Si vemos con un poco más de detalle la temporización de las diferentes señales de sincronización, tenemos que el **reloj del muestreo del monitor funciona a 25'175 MHz** (con lo que toma una muestra de color cada **39'72 ns**). A esta velocidad, para reproducir una línea completa que se compone de 800 píxeles (640 visibles) obtenemos una **frecuencia de sincronización horizontal de 31'469 KHz** y, en consecuencia, para reproducir una imagen completa, compuesta por 525 líneas (480 visibles), obtenemos una **frecuencia de sincronización vertical de 59'94 Hz**. Es decir, 60 imágenes por segundo, que es lo que estábamos buscando.

Hay que tener en cuenta que cuando hablamos de una resolución de 640x480, nos estamos refiriendo únicamente a la parte visible, pero que tenemos que tener en cuenta: el tiempo necesario para estas señales de sincronización, los bordes de la imagen, etc. De ahí que a la hora de hacer los cálculos de las frecuencia, hayamos tomado unos valores de 800x600.

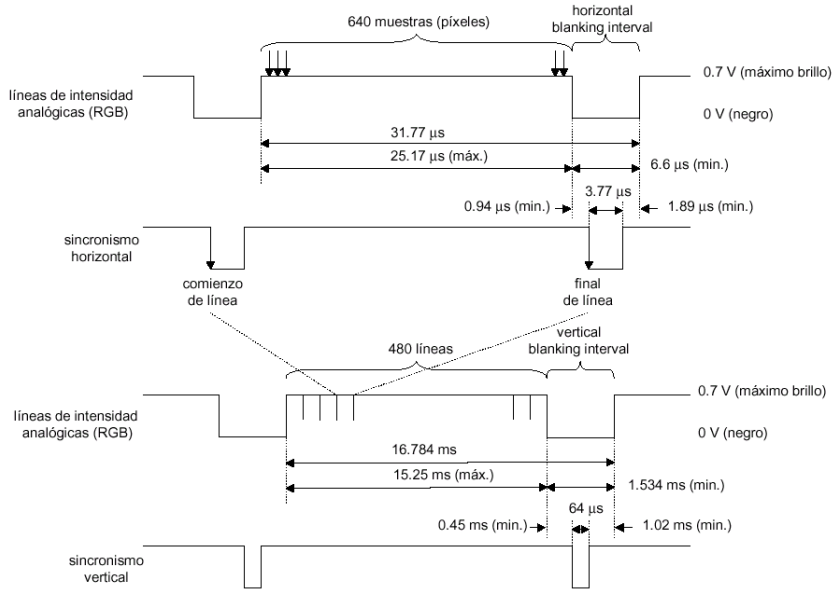
Todo esto queda mucho más claro si vemos una pequeña tabla explicativa en la que tenemos representado cada uno de estos valores en número de píxeles si estamos considerando el movimiento en horizontal y en número de líneas para el caso vertical.

	Porche Delantero	Sincronización	Porche Trasero	Borde Anterior	**Zona Visible**	Borde Posterior	**Total**
Píxeles	8	96	40	8	**640**	8	**800**
Líneas	2	2	25	8	**480**	8	**600**





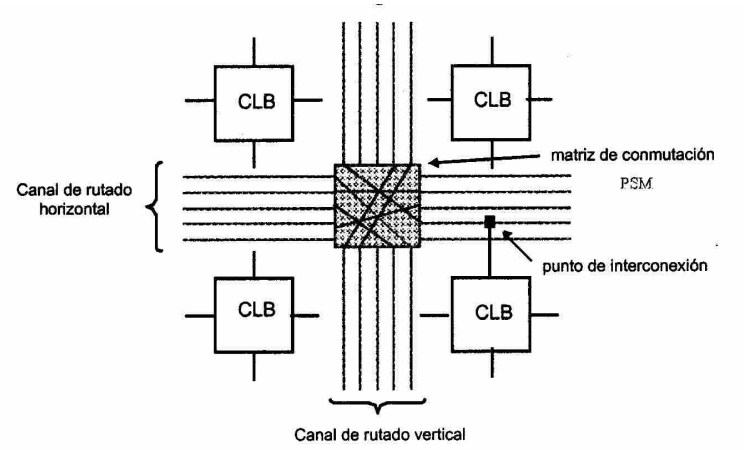
Si ahora consideramos todo esto desde el punto de vista temporal, obtendríamos unos valores como los que aparecen representados en la siguiente figura y que nos dan la temporización definitiva para el sincronismo horizontal y vertical respectivamente.



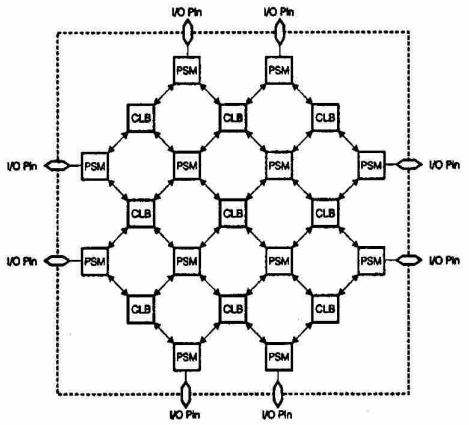
2.2. FPGA

Las FPGA son dispositivos genéricos que pueden configurarse para que implementen un sistema digital en particular. Se utilizan comúnmente en la construcción de prototipos dentro del ciclo de diseño y para aplicaciones de baja tirada.

En las FPGA pueden distinguirse componentes con distinta finalidad. Los **CLB** son las celdas con funcionalidad propiamente reconfigurable, y se hallan dispuestas regularmente por toda la superficie del circuito. Las conexiones con otros circuitos exteriores se realizan mediante las celdas **IOB**, dispuestas por todo el perímetro. Los **PSM** son bloques que pueden programarse para definir las conexiones de celdas CLB entre sí y con los IOB.

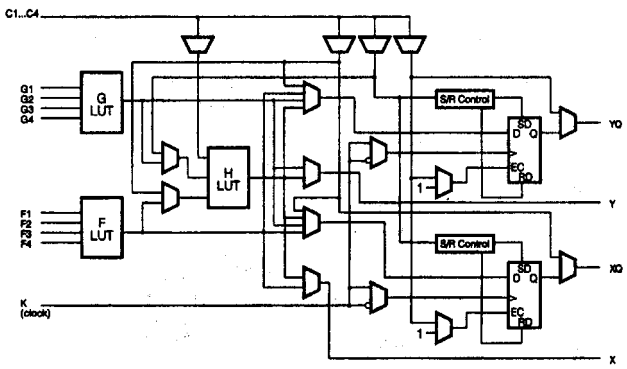


Interconexiones

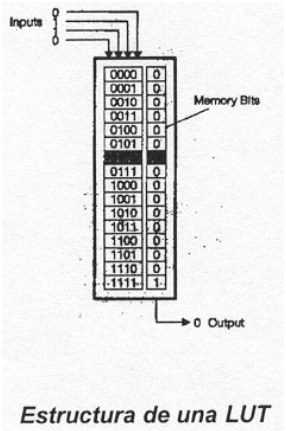


Esquema general

El principio de funcionamiento del CLB está en la selección de qué funciones de conmutación se implementan. Los CLB están compuestos internamente por bloques **LUT** (Look-Up Table), que pueden asimilarse a la tabla de verdad de una función de un número determinado de entradas. También comprenden elementos biestables para cálculos secuenciales. La programación consiste en asignar los valores que corresponden a estas tablas de verdad. El método usado para la programación (RAM, EPROM, ...) determina si el diseño proyectado sobre la FPGA es o no volátil.



Estructura de un CLB



Estructura de una LUT

La composición de elementos de lógica simples para obtener funcionalidades complejas se consigue comunicándolos entre sí, mediante la red de interconexiones. Los PSM determinan qué caminos se habilitan en cada caso para el flujo de las señales de entrada y salida de los CLB.

Por lo tanto, la implementación de circuitos diferentes se logra mediante la reconfiguración de las conexiones internas del circuito y la asignación de valores a las funciones lógicas básicas.

La generalidad es la principal ventaja de estos dispositivos, cuyo diseño y fabricación no dependen de un tipo de diseño en particular. La principal limitación es la capacidad efectiva del circuito, ya que la mayor parte de la superficie se destina a la red de conexiones variables. Así , la complejidad de los diseños que pueden implementarse está restringida. En realidad, en el mercado existen distintas familias de dispositivos FPGA, de capacidad variable y orientadas a campos de aplicación diferentes (los modelos Virtex van aproximadamente de las 50.000 al millón de puertas).

El proceso para el usuario es sencillo, una vez que se tiene la especificación del hardware, por ejemplo en vhdl, existen herramientas específicas para cada tipo de placa que determinan automáticamente la configuración oportuna de las conexiones.

2.3. Placa Virtex XSV-800

La XSV es una placa de prototipado de la familia Virtex que contiene los siguientes componentes:

- Xilinx Virtex FPGA XCV800
- Xilinx CPLD XC65108
- Un oscilador programable Dallas DS1075 que proporciona la señal de reloj a la FPGA y CPLD. Este reloj tiene una frecuencia base de 100 MHz.
- 16 Mbit de memoria Flash para almacenar diferentes configuraciones o datos de propósito general para la FPGA. La memoria es Intel 28F016S5
- Dos bancos de memoria RAM de 512K x 16 bits. Cada banco es formado por dos AS7C4096 SRAM.
- Video - decoder SAA7113 que puede recibir señales NTSC/PAL/SECAM.
- RamDac BT481A que genera las señales para un monitor.
- Stereo- CoDec AK4520A que permite generar y recibir señales de audio.
- Puerto para conexión a una red Ethernet modelo LXT870A.
- Switches.
- Dos displays de siete segmentos y una barra de leds.
- Puerto para la conexión a un teclado o ratón.
- Puerto USB PDIUSBP11A.
- Puerto paralelo.

En la siguiente figura se muestra la arquitectura de la placa:

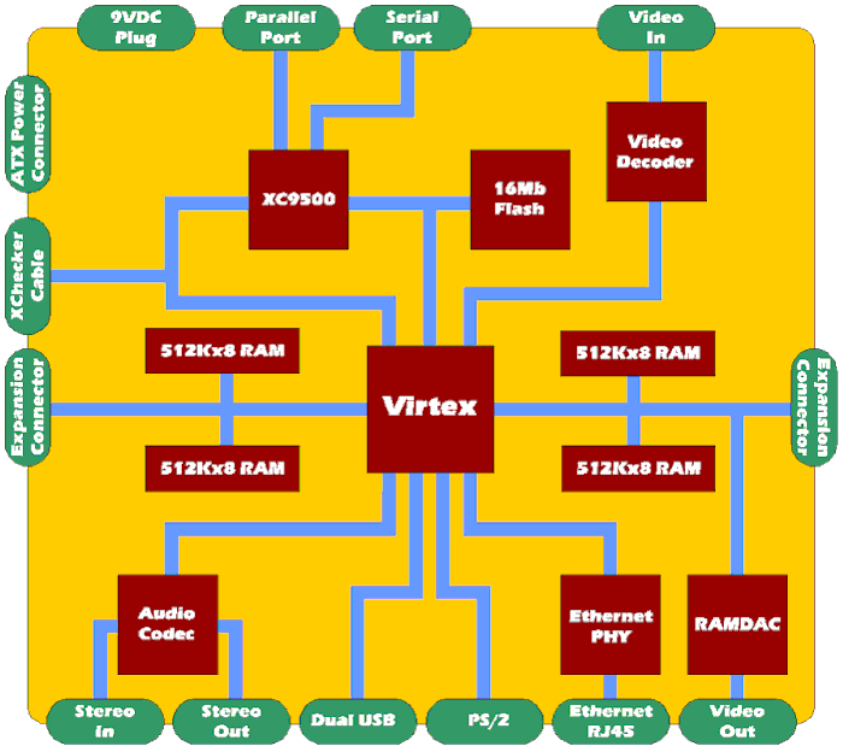


Diagrama de módulos de la XSV 800

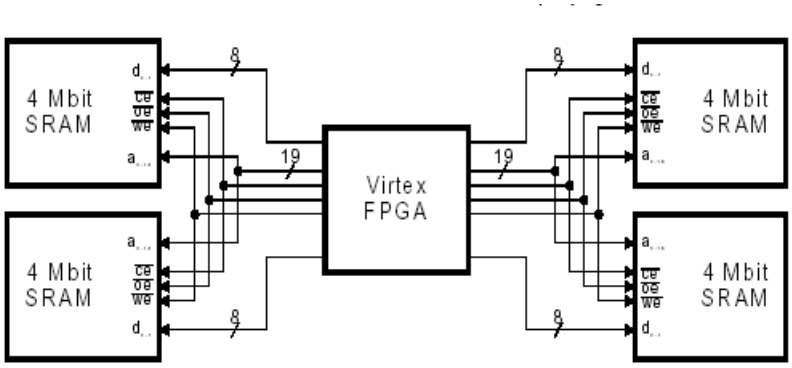
2.3.1. Descripción de los Componentes Utilizados

De los componentes que hay en la placa hemos usado los siguientes:

- Bancos de memoria
- RamDac
- Puerto PS/2
- Puerto paralelo
- Switches

2.3.1.1. Bancos de Memoria

La FPGA tiene acceso a dos bancos de SRAM independientes. Cada banco de la SRAM tiene 512K x 16 bits y está organizado en dos chips de 512K x 8 bits cada uno. En la figura se puede ver la conexión de estos dos bancos con la FPGA.



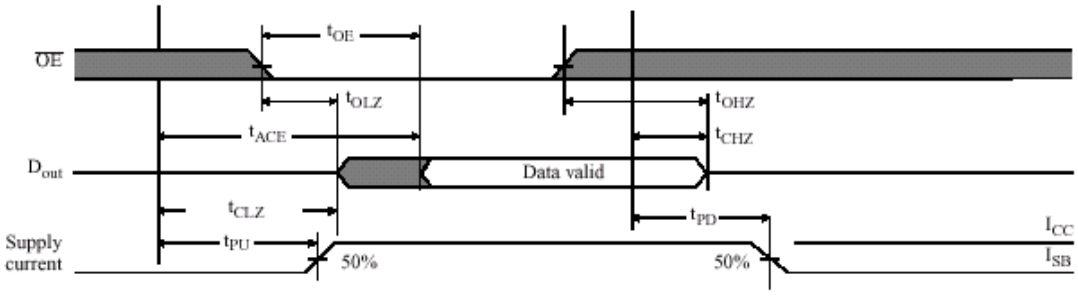
Conexión de los bancos de memoria con la FPGA

Cada banco tiene 19 líneas de dirección y 16 líneas de datos(8 en cada chip), además de las señales de Write Enable (WE), Output Enable (OE) y Chip Enable (CE) todas ellas activas a baja. La señal de WE es prioritaria sobre la de OE.

En el proyecto se utilizan los dos bancos. Cabe destacar que las líneas de datos de la parte baja del banco derecho de memoria están compartidas con el RamDac.

En nuestro caso, el modo de lectura está dirigido por OE y el de escritura por WE, es decir, CE siempre está activo.

El esquema de temporización de la lectura es el siguiente:



Temporización de la lectura

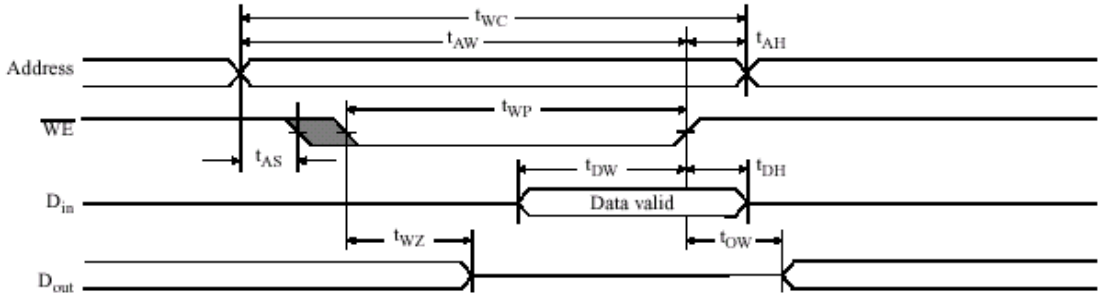
La tabla de los tiempos de lectura son los siguientes:

Parameter	Symbol	-15 Min	-15 Max	-20 Min	-20 Max	-25 Min	-25 Max	Unit	Notes
Read cycle time	t_{RC}	15	–	20	–	25	–	ns	
Address access time	t_{AA}	–	15	–	20	–	25	ns	3
Chip enable ($\overline{CE}$) access time	t_{ACE}	–	15	–	20	–	25	ns	3
Output enable ($\overline{OE}$) access time	t_{OE}	–	4	–	5	–	6	ns	
Output hold from address change	t_{OH}	3	–	3	–	3	–	ns	5
$\overline{CE}$ LOW to output in Low Z	t_{CLZ}	0	–	0	–	0	–	ns	4, 5
$\overline{CE}$ HIGH to output in High-Z	t_{CHZ}	–	4	–	5	–	6	ns	4, 5
$\overline{OE}$ LOW to output in Low Z	t_{OLZ}	0	–	0	–	0	–	ns	4, 5
$\overline{OE}$ HIGH to output in High-Z	t_{OHZ}	–	4	–	5	–	6	ns	4, 5
Power up time	t_{PU}	0	–	0	–	0	–	ns	4, 5
Power down time	t_{PD}	–	15	–	20	–	25	ns	4, 5

Shaded areas contain advance information.

Tabla de tiempos de lectura

El esquema de temporización de la escritura es el siguiente:



Temporización de la escritura

Las tabla de tiempos de escritura están descritos en la siguiente tabla:

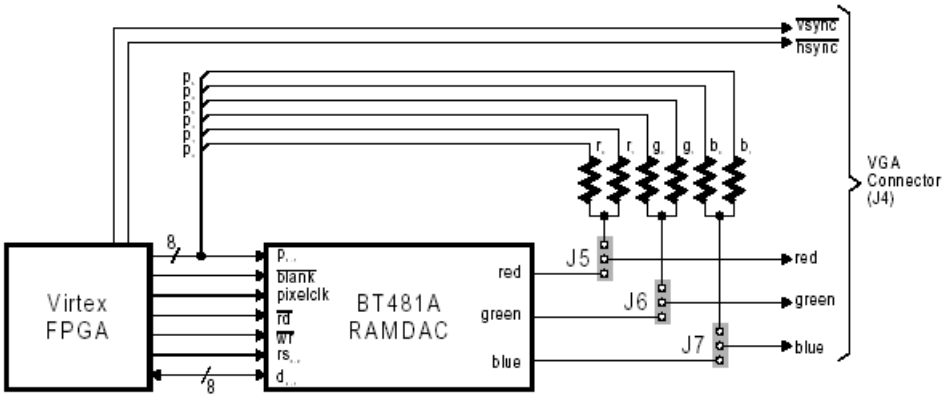
Parameter	Symbol	-15 Min	-15 Max	-20 Min	-20 Max	-25 Min	-25 Max	Unit	Notes
Write cycle time	t_{WC}	15	–	20	–	20	–	ns	
Chip enable ($\overline{CE}$) to write end	t_{CW}	12	–	12	–	15	–	ns	
Address setup to write end	t_{AW}	12	–	12	–	15	–	ns	
Address setup time	t_{AS}	0	–	0	–	0	–	ns	
Write pulse width	t_{WP}	9	–	12	–	15	–	ns	
Address hold from end of write	t_{AH}	0	–	0	–	0	–	ns	
Data valid to write end	t_{DW}	9	–	10	–	10	–	ns	
Data hold time	t_{DH}	0	–	0	–	0	–	ns	4, 5
Write enable to output in High-Z	t_{WZ}	–	5	–	5	–	5	ns	4, 5
Output active from write end	t_{OW}	3	–	3	–	3	–	ns	4, 5

Shaded areas contain advance information.

Tabla de tiempos de la escritura

2.3.1.2. RamDac

RamDac (Random Access Memory Digital-to-Analog Converter) convierte una señal digital en una señal analógica que es la que necesita el monitor. En la siguiente figura se muestra la conexión del RamDac con la FPGA.

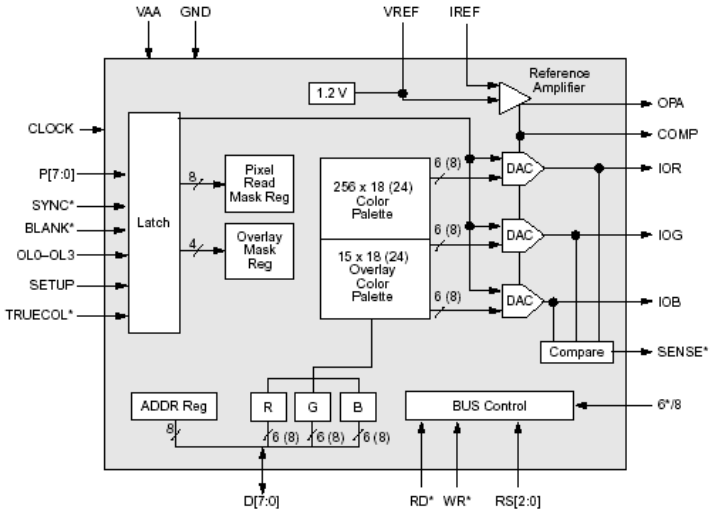


Conexión del RamDac con la Virtex

El RamDac tiene una pequeña RAM que se usa como paleta de color y tres conversores digitales analógicos (DAC) que convierten la señal digital en señales analógicas. Hay un DAC por cada color : rojo, verde y azul. El RamDac tiene dos modos de funcionamiento programable mediante la FPGA:

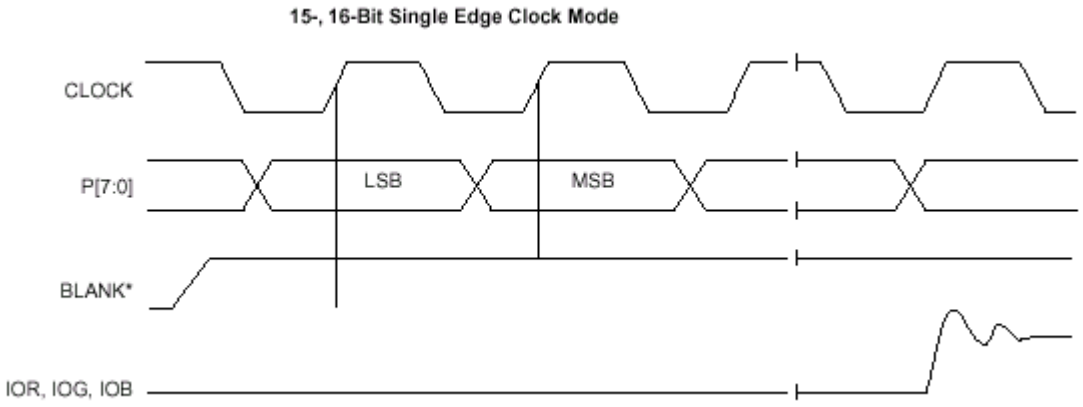
- Pseudo color: cuando el RamDac funciona en este modo lee un color de la paleta y a partir de el se generan tres valores separados para cada una de las componentes RGB. Estos valores van directamente al cañón de electrones del monitor.
- True color: en este modo el dato digital pasa directamente a los DAC. En trae color hay tres modos de funcionamiento:
    - 5:5:5 (15 bits, 32K colores)
    - 5:6:5 (16 bits, 65K colores)
    - 8:8:8 RGB (24 bits, 16.8M colores)

En la siguiente figura se puede ver la arquitectura del RamDac.



El RamDac se ha programado en true color en el formato 5:5:5 con un solo flanco. Para la programación del RamDac, el registro de comando A en true color tiene que tomar el valor "10100000" y como sólo accedemos al registro de comando A, el registro RS (Register Select) tiene el valor "110".

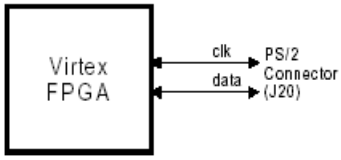
Una vez programado el RamDac se va leyendo en cada flanco de subida del reloj del RamDac primero la parte menos significativa y a continuación la más significativa de cada dato de 16 bits. El esquema es el siguiente:



Temporización del RamDac

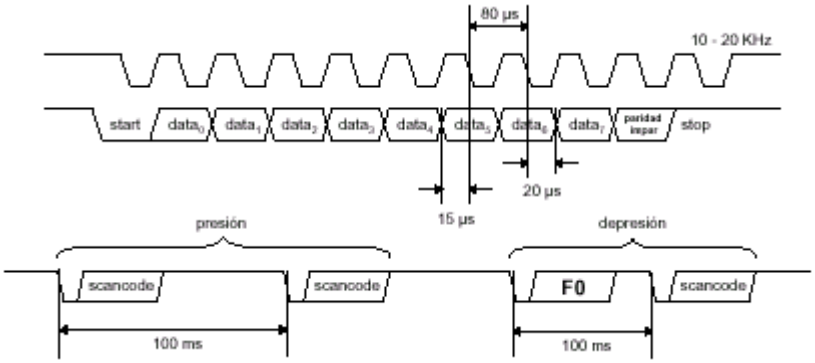
2.3.1.3. Puerto PS/2

La FPGA tiene un interfaz para un teclado o un ratón. La FPGA recibe dos señales de la interfaz del PS/2: la señal de reloj y una línea de datos serie sincronizado con los flancos de bajada de la señal de reloj.



Conexión del teclado con la Virtex

El dispositivo para transmitir un dato pone la señal de data a baja. El sistema muestrea en cada flanco de bajada de reloj, por lo que los datos se deben fijar antes. En la siguiente figura se puede ver la temporización de un interfaz de teclado:



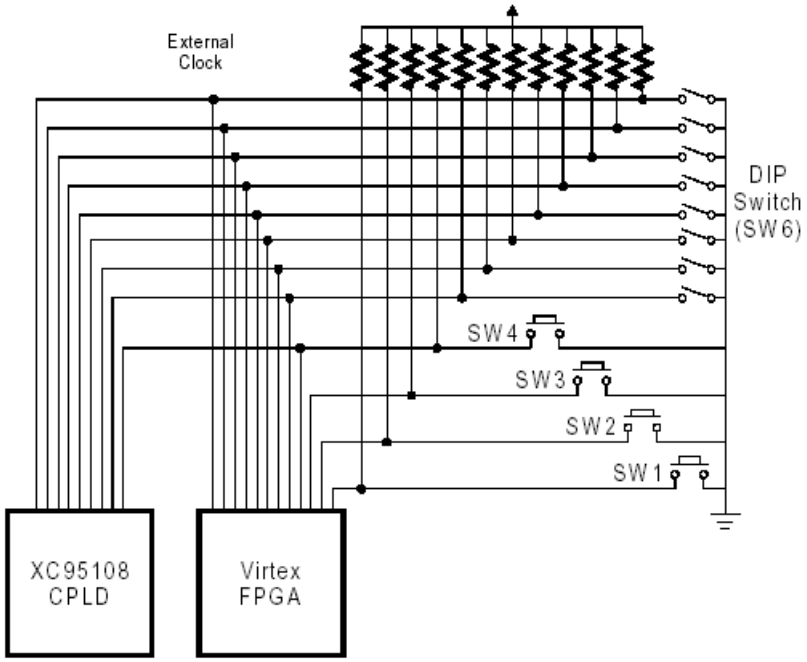
Temporización del teclado

El teclado se utiliza para seleccionar los diferentes efectos implementados, así como para modificar los modos de funcionamiento de éstos.

2.3.1.4. Switches

La Virtex tiene un banco de ocho switches DIP y cuatro botones accesibles en la propia FPGA.

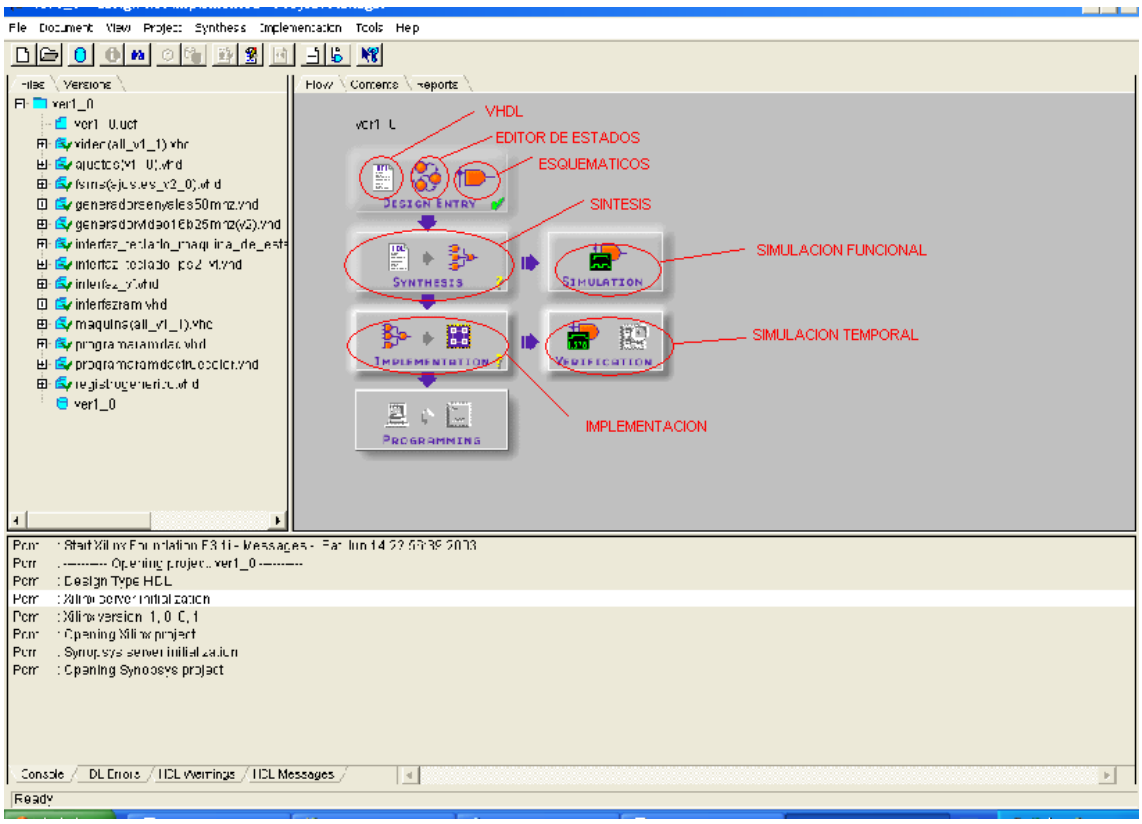
En la versión final del proyecto tan sólo se usa un switch (botón) para el reset del sistema (activo a baja), sin embargo los botones han sido de gran utilidad durante el desarrollo del mismo.



Conexión de los switches con la Virtex

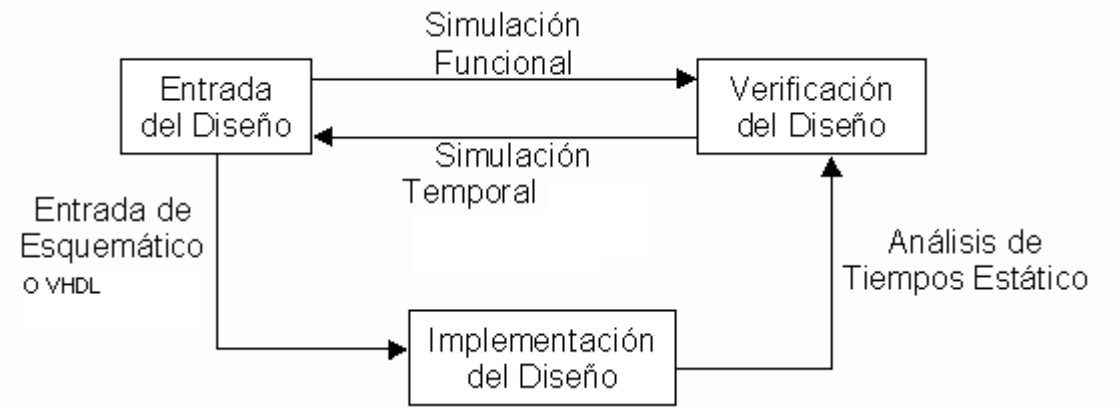
2.4. Xilinx Foundation

Xilinx Foundation es una herramienta de desarrollo que consiste en un conjunto integrado de herramientas software y hardware para crear, simular e implementar diseños digitales en una FPGA o CPLD. Todas las herramientas usan una interfaz de usuario gráfica que permite usar todos los programas desde iconos, menús o barras de herramientas. También se dispone de ayuda en línea desde la mayoría de ventanas.

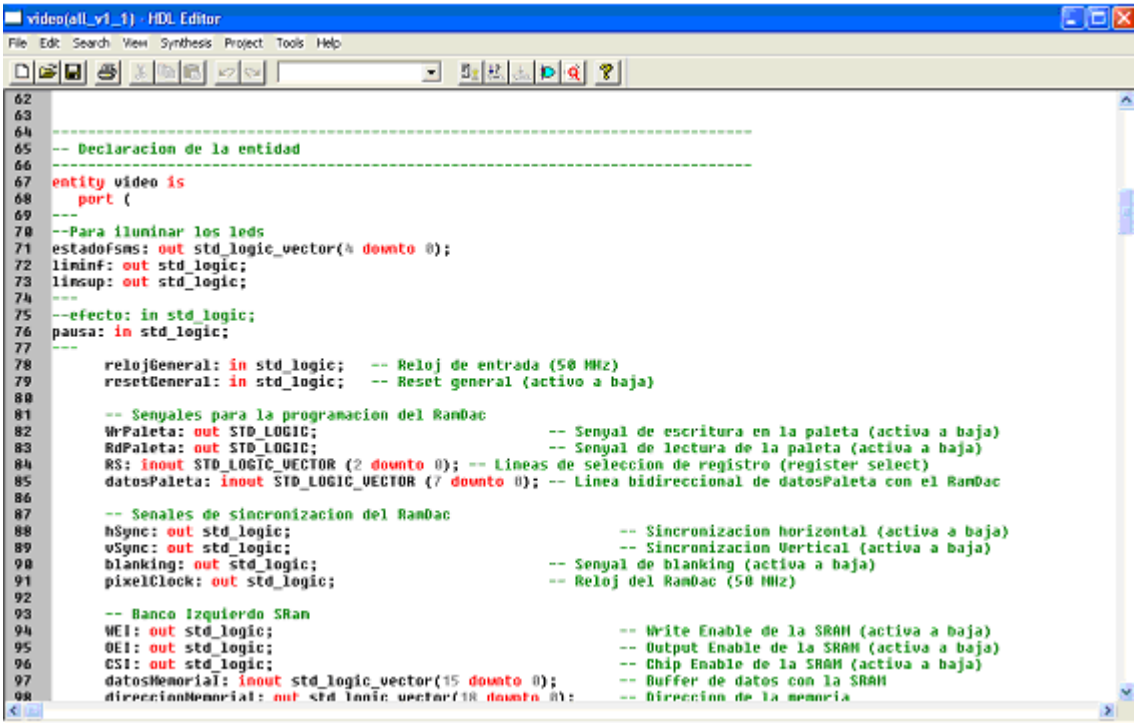


Entorno gráfico de Xilinx Foundation

El proceso de diseño consta de los siguientes pasos :



Entrada del diseño: el diseño se puede introducir como un esquemático o mediante un lenguaje de descripción hardware que permite describir funcionalmente el circuito o una combinación de ambas .



Fichero VHDL

Implementación del diseño: Las herramientas de implementación sintetizan el diseño realizado en la arquitectura del dispositivo seleccionado. Las herramientas de implementación están automatizadas y compilan el diseño en un fichero de configuración que es el que se usa para programar el dispositivo real, optimizado en términos de uso de puertas lógicas e interconexiones para el dispositivo dado.

El proceso de síntesis de un FPGA consta de las siguientes etapas:

a) ***Translate***: convierte el archivo de diseño EDIF en un archivo NGD.

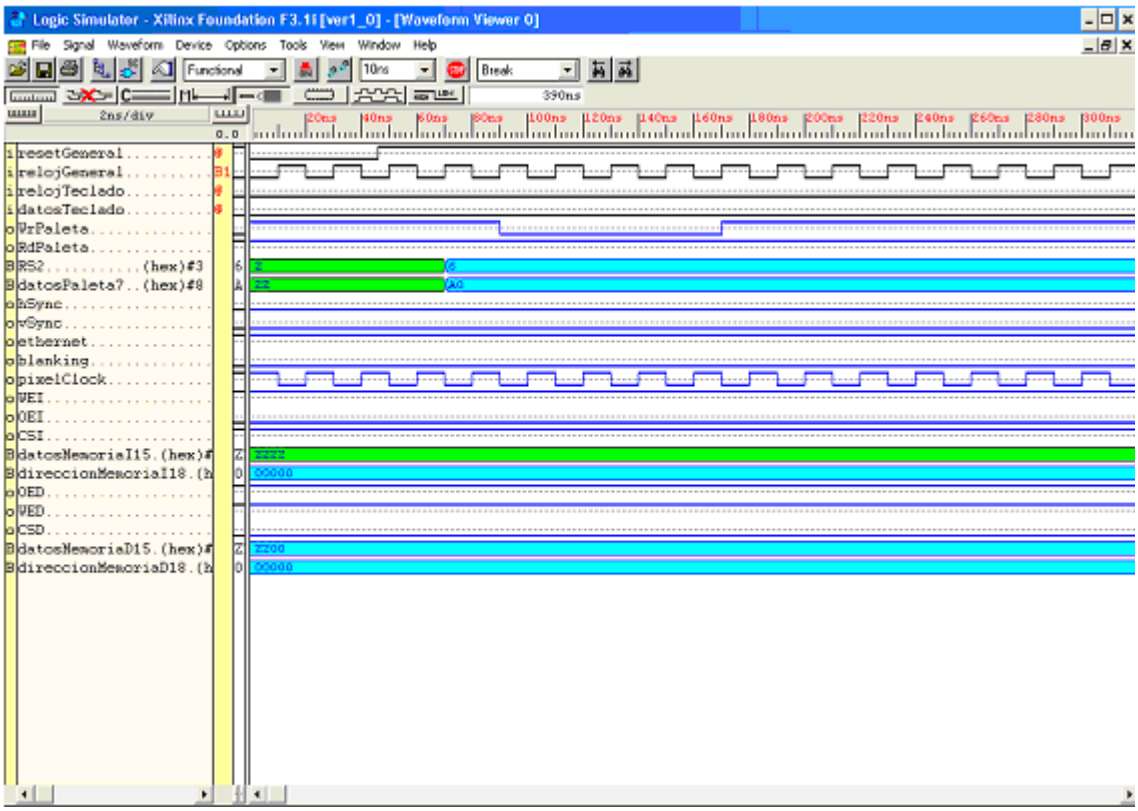
b) ***Map***: agrupa los elementos básicos (flip–flops, compuertas, etc.) en bloques lógicos y genera el reporte de tiempo a nivel lógico.

c) ***Place&Route***: localiza los bloques lógicos en el dispositivo y canaliza las señales entre ellos.

d) ***Timing***: genera datos de simulación de tiempo y el reporte de tiempo de *post-layout*.

e) ***Configure***: genera el archivo (.BIT o .SVF) adecuado para programar al dispositivo. Con la ejecución de estas etapas el proceso de implementación termina. En caso de existir errores en alguna etapa, el proceso de implementación se detiene y se genera un reporte indicando la causa.

Verificación del circuito : incluye la simulación funcional, que se realiza sobre el ordenador, el testeo del circuito, que se realiza tras su programación, y la simulación temporal (una vez rutado el dispositivo).



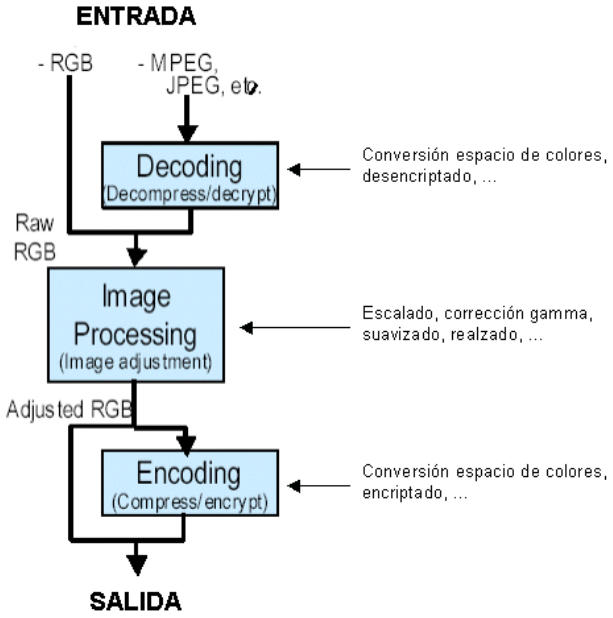
Verificación Funcional

Al finalizar el proceso de síntesis la herramienta habrá generado el archivo con extensión .bit para la programación del FPGA, que se efectúa a través de la herramienta **XSTOOLS**, la cual establece comunicación entre el PC y la tarjeta vía puerto paralelo.

3. El Sistema

3.1. Descripción del Escenario

El proceso general del tratamiento de imágenes digitales podemos dividirlo en tres fases: **adquisición**, **procesado** y **reproducción**, como queda reflejado en la siguiente figura:



La **adquisición** consiste en la captura de la imagen, que puede ser realizada por infinidad de componentes, desde una cámara de video hasta un escáner. En determinados casos, será necesario un proceso de decodificación adicional, debido a que estas imágenes pueden estar codificadas mediante alguno de los algoritmos de compresión de video que se han adoptado como estándar, como MPEG, JPEG o DIVX que están imponiéndose a causa de las grandes demandas en cuanto a las capacidades de transmisión de imágenes y contenidos multimedia, en general a través de la red.

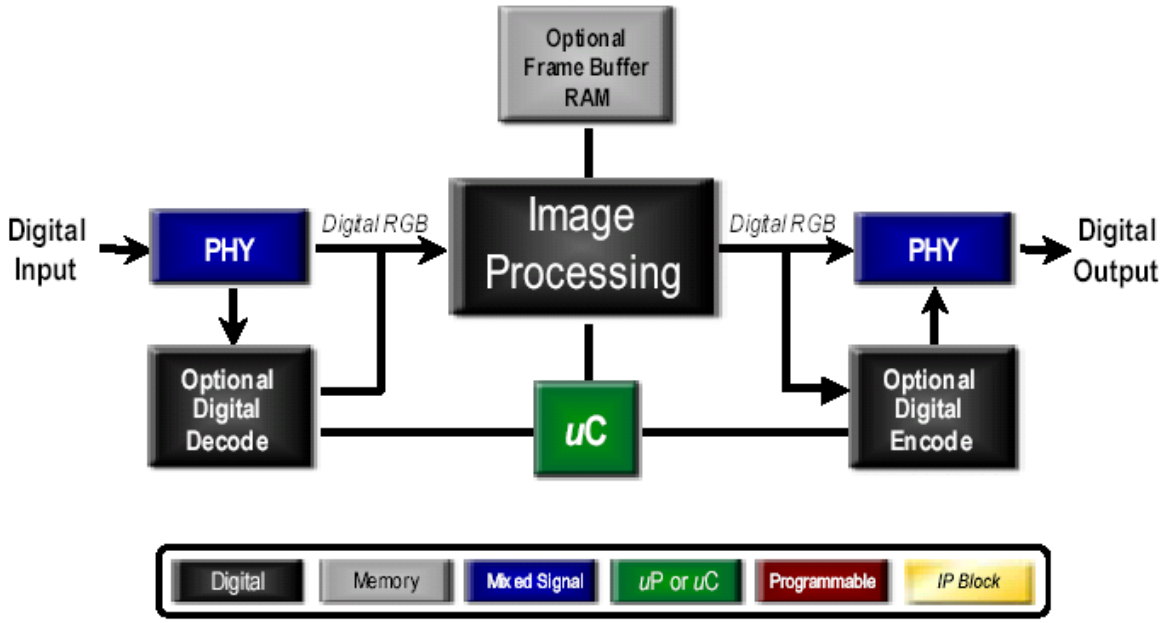
Si tratáramos de transmitir estas imágenes tal y como han sido adquiridas, en formato RGB, los requisitos de ancho de banda y capacidad de almacenamiento entre otros, serían muy elevados. Para solucionarlo, se realiza un paso intermedio de conversión de espacio de colores en el que convertimos estas imágenes a un formato con requisitos mucho menores: Y, Cb, Cr por lo que aquí será necesario convertir estas señales en el espacio de colores para que podamos tratarlas convenientemente.

A continuación, la imagen se **procesa** realizando todas aquellas operaciones que creamos oportunas. Generalmente suelen aplicarse tres fases: un primer paso encaminado a mejorar la calidad o a reducir los fallos en la calidad imagen; en un segundo paso, los efectos en sí; y un tercero en el que vuelve a ajustarse la imagen para volver a reducir al mínimo los posibles fallos que hayan podido volver a surgir.

Finalmente, el último paso será la **reproducción** de la imagen que dependerá en gran medida del dispositivo sobre el que la vayamos a mostrar. En este caso, también puede ser necesario un proceso de compresión con alguno de los algoritmos mencionados anteriormente. En cualquier caso, aquí será necesario una nueva conversión en el espacio de colores, inverso al que realizamos previamente.

3.1.1. Hardware y Tratamiento de Imágenes

Si extrapolamos el flujo que se sigue en el tratamiento de las imágenes digitales que vimos en el apartado anterior, y tratamos de ver qué hardware está implicado en cada una de las fases, obtendríamos una figura similar a la siguiente:



Las señales de entrada han de ser convertidas del espacio de colores de Y, Cb, Cr a RGB porque el procesado lo realizamos en este último. Tanto para las señales de entrada como para las de salida, puede ser necesario un proceso adicional de decodificación o codificación respectivamente.

En cualquiera de los dos casos, ambos procesos dependen directamente del controlador del que dispongamos (µC) y que será el encargado de gestionar todo el procesado de la imagen (caja central).

Generalmente, será necesario almacenar temporalmente las imágenes en lo que se llama frame buffer para poder procesarlas adecuadamente, y en la mayoría de los casos, requerirá el uso de memorias RAM.

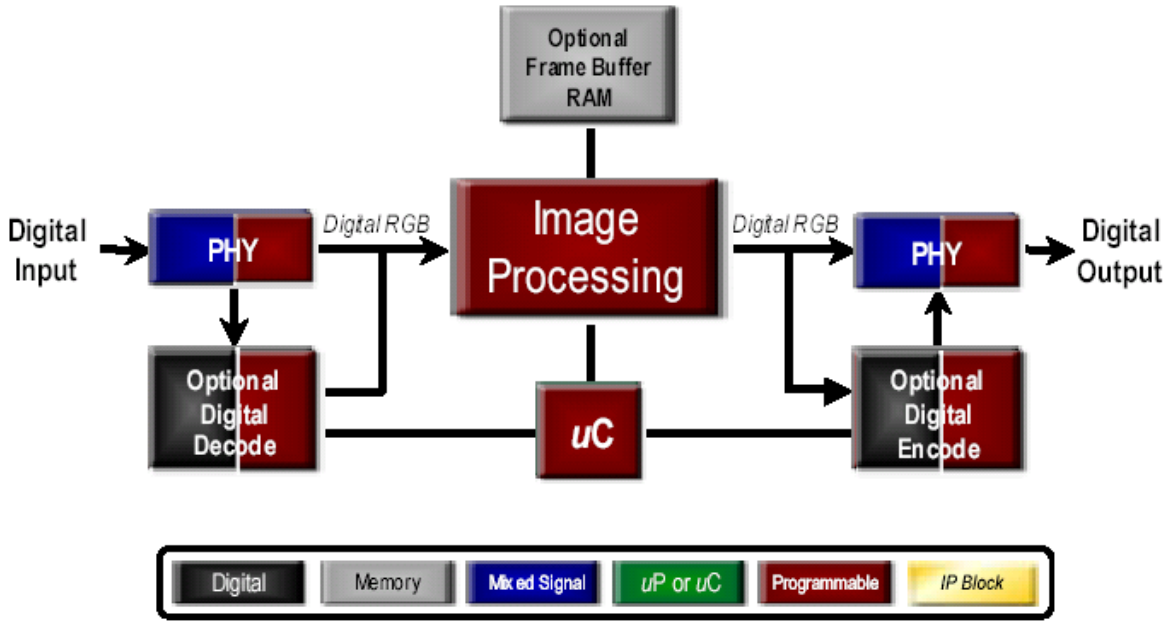
Una vez procesada la imagen, el último paso será la reproducción de la misma. La reproducción de la imagen puede requerir una codificación previa (en el esquema representada por la caja *Optional Digital Encode*).

3.1.2. La Utilidad de las FPGA en el Tratamiento de Imágenes

Se ha demostrado que en el tratamiento de imágenes digitales se obtienen rendimientos mucho mayores a todos los niveles cuando el proceso se realiza directamente mediante hardware, en lugar de utilizar software que lo emule.

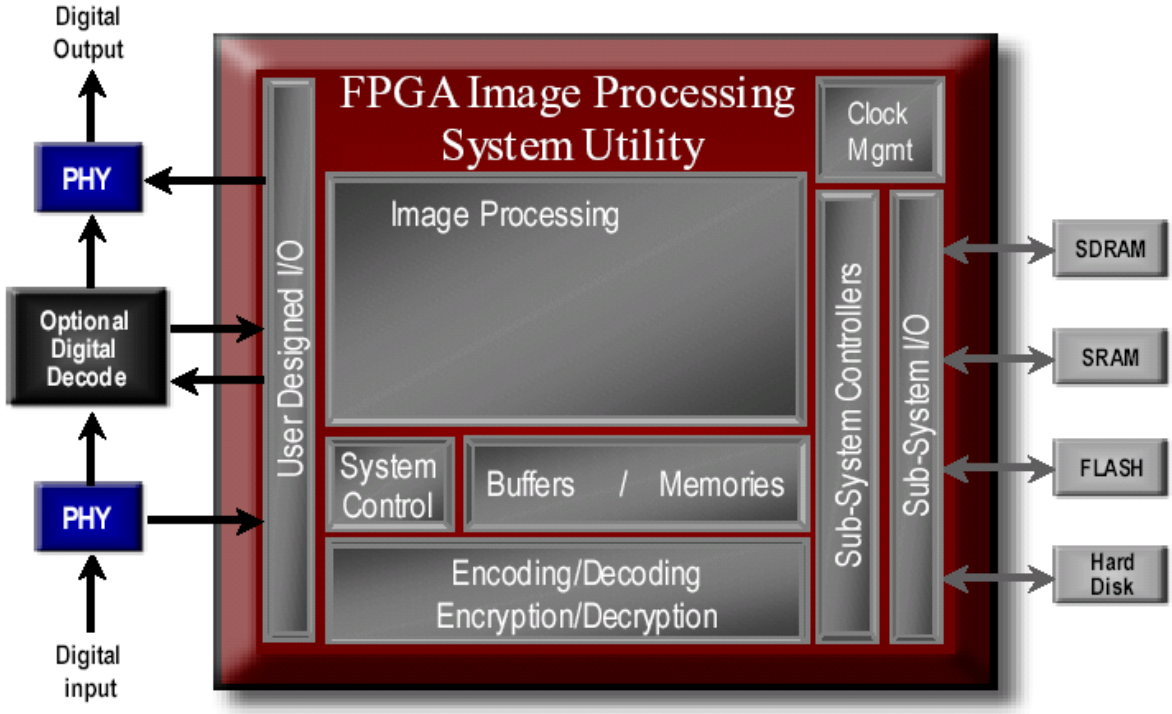
En esto, desempeña un papel muy importante la lógica programable, y por extensión las FPGA, que permiten alcanzar todas las ventajas del uso de este hardware específico, pero aportando además las ventajas inherentes a la reconfiguración, por lo que pueden utilizarse según demanda siempre que el componente pueda satisfacer los requisitos exigidos.

Si representamos sobre el gráfico anterior aquellas áreas en las que las FPGA (la lógica programable) juegan un papel especialmente determinante, obtendríamos una figura similar a la siguiente:



Como vemos, las FPGA abarcan por completo la unidad de procesado de la imagen y el controlador, una parte de la adquisición (en la que se realiza la conversión en el espacio de colores) y una parte de la reproducción (en la que volvemos a hacer la conversión en el espacio de colores).

Representando dicha unidad sobre una FPGA obtenemos un esquema como el siguiente:



Como podemos observar, la FPGA se va a encargar de sustituir a la mayor parte del hardware que se requiere para el tratamiento de las imágenes así como de la programación de algunos dispositivos e interfaces.

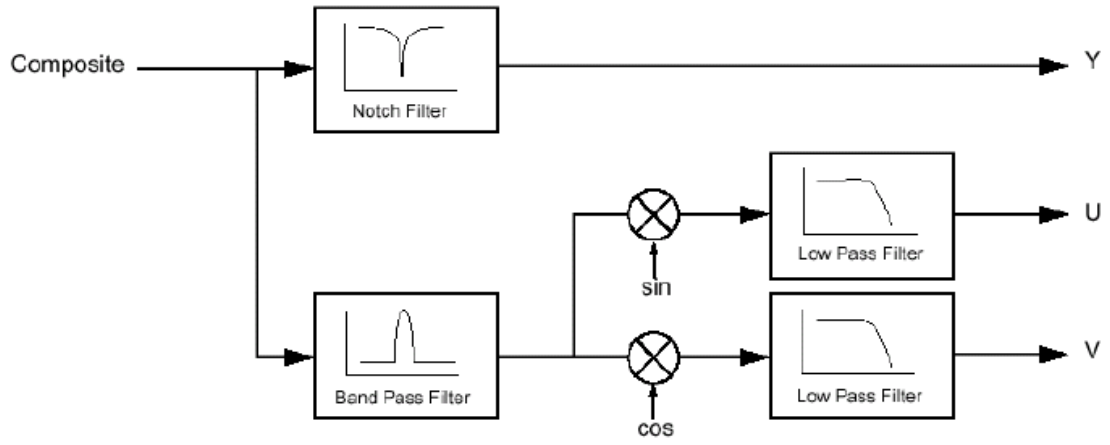
3.1.3. Conversión entre Espacios de Colores

3.1.3.1. Conversión de Vídeo Compuesto a YUV

El paso inmediatamente posterior a la adquisición de las imágenes es su conversión al espacio de colores adecuado para que su transmisión requiera los mínimos recursos posibles, conservando la calidad de la imagen original.

Como ya hemos visto, esto se suele hacer en el espacio de colores YUV mediante la aplicación de una serie de filtros que nos permiten extraer las tres componentes.

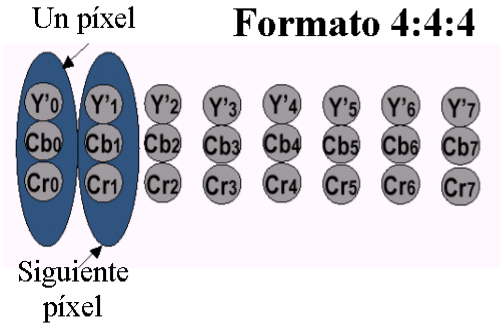
El esquema general de la extracción sería similar al siguiente:

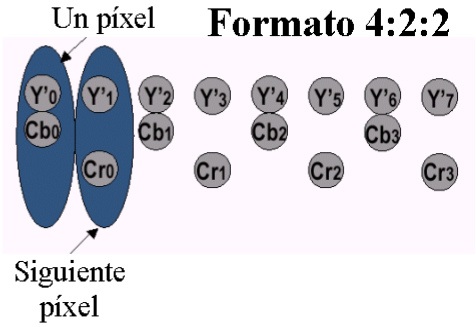


De esta tarea generalmente se encarga un componente específico que ha de conectarse al circuito que posteriormente procesará la imagen adquirida, por lo que no entraremos a profundizar más en él.

Aunque hay varias formas de transmitir esta información, las más habituales son las llamadas 4:4:4 y 4:2:2. Estos números nos indican la relación que hay entre la cantidad de cada una de las componentes que se están transmitiendo. Así, para el caso de 4:2:2, tenemos dos muestras U y dos muestras V por cada 4 muestras de luminancia Y.

Gráficamente lo vemos mucho mejor:





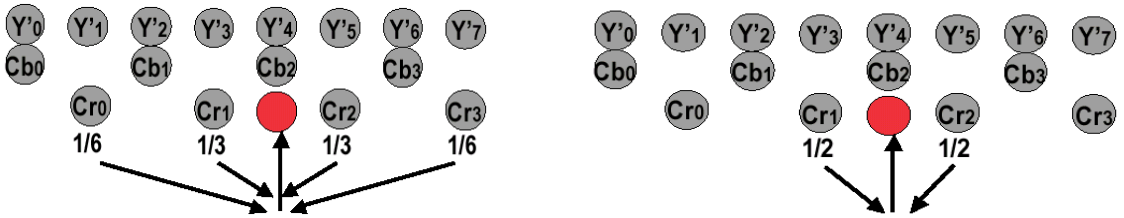
3.1.3.2. Conversión desde YUV a RGB

Esta conversión sí se implementa en el propio circuito de procesado y está basada en una serie de relaciones matemáticas entre las componentes YUV y las componentes RGB.

Hay muchas implementaciones diferentes en función, tanto de los coeficientes utilizados en estas relaciones (que determinarán la pérdida de calidad y la bondad de la conversión), como del tipo de formato YUV que estemos usando.

Lo más usual es partir de una codificación en el espacio YUV con el formato 4:2:2 que acabamos de ver, en la que tenemos dos muestras U y dos muestras V por cada cuatro de luminancia Y (el doble).

Por lo tanto, para recuperar el formato original (4:4:4) a partir de éste, intuitivamente, vemos que podemos adoptar dos aproximaciones en función de si queremos una mejor calidad con unos requisitos también mayores (la de la izquierda) o si por el contrario, no necesitamos esas cotas de calidad o no disponemos de las capacidades de procesamiento requeridas (la de la derecha).



Es decir, estamos interpolando las componentes de crominancia.

La relación entre cada una de las componentes y el píxel con el que se corresponden es la siguiente:

para n=0, 2, 4:

$$Cb_n = Cb_n$$
$$Cr_n = Cr_n$$
$$Cb_{n+1} = (Cb_n + Cb_{n+2}) / 2$$
$$Cr_{n+1} = (Cb_n + Cb_{n+2}) / 2$$

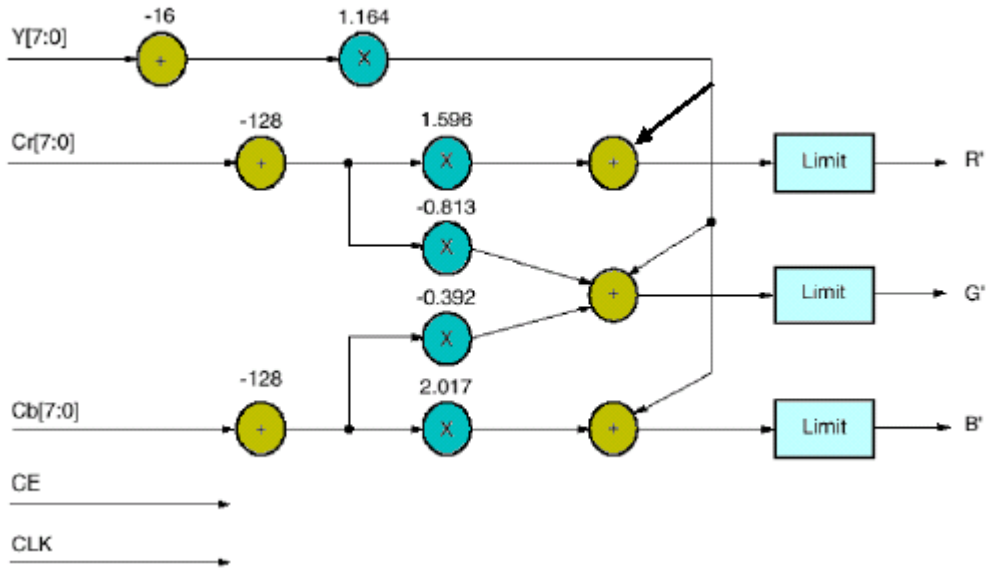
Es decir, las dos componentes de crominancia de un píxel se conservan igual, mientras que la del siguiente es la media ponderada del anterior y el posterior.

Si profundizamos un poco más en el circuito necesario para esta conversión, vemos que, si queremos que la conversión sea lo suficientemente buena y así podamos recuperar la imagen original, no nos sirve con estas aproximaciones, sino que tenemos que ajustar más los coeficientes. Obviamente, esto sólo será posible si disponemos de un procesador lo suficientemente rápido como para poder soportar esta sobrecarga en los cálculos.

Las ecuaciones estándar aplicadas para la conversión a RGB son:

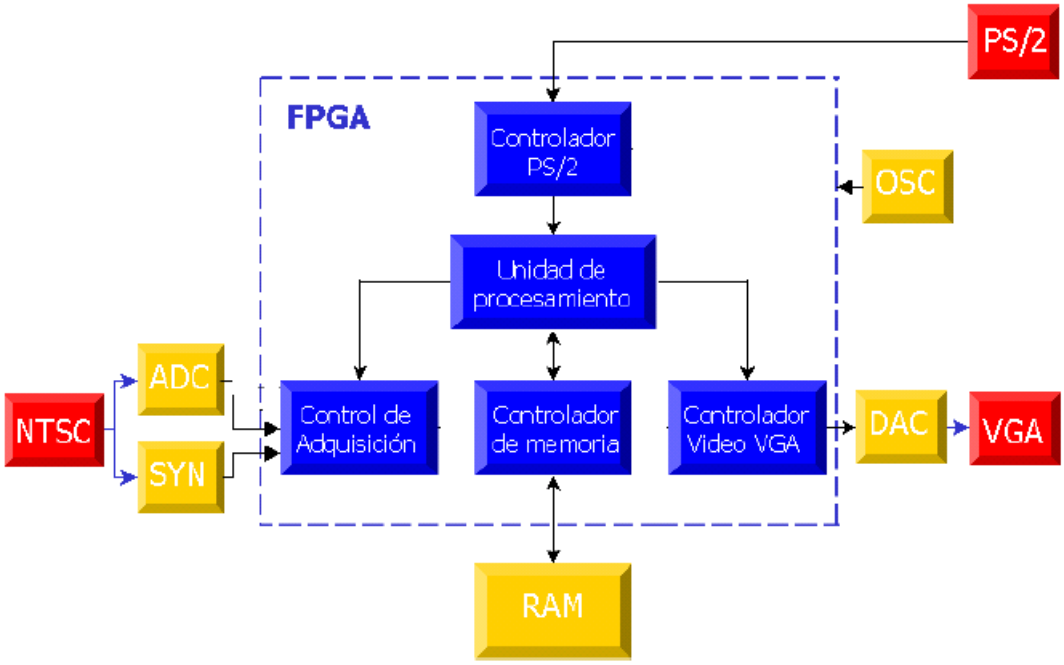
R = 1.164 (Y – 16) + 1.596 (Cr – 128)	Y ∈ [16, 235]
G = 1.164 (Y – 16) – 0.813 (Cr – 128) – 0.391 (Cb – 128)	Cr / Cb ∈ [16, 240]
B = 1.164 (Y – 16) + 2.018 (Cb – 128)	RGB ∈ [0, 255]

Si analizamos los requisitos que esto nos supone, vemos que se reduce a disponer de un circuito que haga una serie de sumas, restas y multiplicaciones, como se ve en la siguiente figura:



3.2. El Sistema en General

El diagrama de bloques de un procesador de imágenes digitales sería similar al que se muestra en la siguiente figura:



Los bloques situados dentro del cuadrado dibujado con líneas discontinuas pertenecen a la FPGA. Fuera tenemos una serie de bloques que conforman los dispositivos. Tenemos dos conjuntos de dispositivos: los dispositivos periféricos como son los conversores, memoria, sincronizador y oscilador (ADC, DAC, RAM, SYN y OSC), y los dispositivos externos tales como el monitor VGA, el teclado y la señal de video (VGA, PS/2 y NTSC).

La señal de video la toma un conversor analógico digital (ADC) y un circuito que extrae los pulsos de sincronismo (SYN). La unidad de adquisición de la FPGA se encarga de decodificar la información que trae consigo e indicarle al controlador de la memoria los instantes en los que debe almacenar un nuevo píxel, la posición de memoria en la que lo debe almacenar y el valor de dicho píxel.

Puesto que nuestro proyecto está enfocado al tratamiento de imágenes digitales, la digitalización en sí queda fuera del mismo, por lo que nosotros hemos optado por simular este comportamiento con un módulo que nos entrega el equivalente a estas muestras ya digitalizadas.

A su vez, el controlador de video VGA se encarga de generar las señales de sincronismo que requiere el monitor y enviarle de forma adecuada los píxeles correspondientes a cada posición de la pantalla.

Debido a que las señales RGB del monitor son analógicas, se requiere un conversor digital-analógico (DAC) de alta velocidad que se encargue de transformar las

señales digitales que entrega el controlador de vídeo en las muestras analógicas que procesará el monitor VGA.

Simultáneamente, la unidad de procesamiento va a estar comunicándose continuamente con la memoria enviándole los datos que están entrando desde el video y que queremos almacenar en ella, y pidiéndole los datos que haya que enviarle al monitor para su muestreo.

El proyecto va a centrarse en el procesamiento de las imágenes que se reciban del módulo que simula el comportamiento del video. La imagen que se ve por el monitor VGA es la que genera el módulo de video. Opcionalmente, el usuario puede elegir el tipo de procesamiento que se realizará sobre la imagen. A los distintos tipos de procesamiento de la imagen los hemos denominado “efectos”.

La selección del efecto se realizará mediante teclado. Cada efecto tiene asignada una tecla distinta y para pasar de uno a otro se deberá cancelar el que se encuentre activo y posteriormente seleccionar el nuevo.

Se requerirá un interfaz PS/2 entre el teclado y el resto del sistema que se encargue de recibir las pulsaciones enviadas por el teclado y recodificarlas de manera que el sistema sea capaz de procesarlas.

Los efectos implementados son los siguientes :

- **Pausa:** congela la imagen en reproducción y la almacena.
- **Memoria:** recupera la última imagen almacenada.
- **Mosaico:** modifica la resolución con que se muestra la imagen, según un factor variable.
- **Zoom:** modifica el tamaño con que se muestra la imagen, según un factor de aumento o reducción variable.
- **Reflejo:** obtiene la imagen especular horizontal o vertical.
- **Rotación:** rota la imagen a lo largo del eje Z un número de grados seleccionable.
- **Wipe:** divide verticalmente la pantalla para mostrar simultáneamente la imagen del vídeo y la última almacenada. El tamaño de la división es ajustable.
- **Pip:** muestra simultáneamente la imagen del vídeo y, en un cuadrante, la última imagen almacenada. El cuadrante se puede seleccionar.
- **Estrobe:** congela periódicamente la imagen. El periodo es ajustable.
- **Estrobe cíclico:** congela periódicamente la imagen, mostrando las cuatro últimas simultáneamente, actualizándolas cíclicamente. El periodo es ajustable.

3.3. Introducción al Diseño

Como hemos visto en la introducción inicial de esta memoria, el objetivo de este proyecto es el análisis, diseño e implementación sobre una FPGA de altas prestaciones (Xilinx XSV-800) de un circuito específico para el procesado digital en tiempo real de una señal de video previamente digitalizada y su visualización sobre un monitor VGA estándar.

Dentro del procesado de la imagen se incluyen, entre otros efectos: la pausa y memorización, efecto estroboscópico, visualización simultánea de varias imágenes efecto zoom y mosaico, rotación e imagen especular, etc.

El primer paso en el desarrollo del proyecto ha consistido en la recopilación de documentación sobre todas las tecnologías relacionadas, que queda resumida en la primera parte de esta memoria. El estudio de esta documentación plantea una serie de alternativas de diseño para las diferentes necesidades, con soluciones no siempre compatibles entre ellas.

El resto de esta sección describe cuáles han sido las decisiones adoptadas finalmente y las razones que las han motivado.

Nuestro sistema general va a funcionar con una **frecuencia de reloj** de 50 MHz, lo que nos da un tiempo de ciclo de 20 ns, por lo que tendremos que adaptar todos los procesos y capacidades del resto de componentes y procesos a esta velocidad, con las consiguientes limitaciones que esto puede suponer.

En cuanto al **color**, teníamos tres opciones principales: escala de grises, pseudo color y color real. Las dos primeras requieren la configuración e inicialización de la paleta del RamDac y comunicaciones internas con un ancho de palabra de 8 bits, que es la cantidad necesaria para codificar la información para cada unos de los píxeles de la imagen. La tercera, requiere únicamente la configuración inicial del RamDac sin inicialización de la paleta. Para la **codificación** de cada píxel de color existen, a su vez, tres alternativas en cuanto al número necesario de bits. Por razones de ancho de banda en la comunicación con la memoria, hemos optado por emplear una codificación de 5 bits (5:5:5) para cada componente de color (RGB), en lugar de, por ejemplo, usar una resolución inferior. Por lo tanto se requieren 16 bits para cada píxel.

La elección del modo de funcionamiento no ha sido arbitraria, se basa en pruebas previas sobre cada uno de ellos. Tras el análisis de los resultados obtenidos y ver que las opciones eran viables, preferimos finalmente trabajar en color real, aunque los requisitos eran mucho mayores. Entendemos que este modo engloba en cierta forma a los otros, por lo que la implementación de estos otros modos resultaría relativamente sencilla a partir del que hemos implementado.

En cuanto a la resolución, entre todos los estándares disponibles, hemos optado por una **VGA estándar**. Esta decisión tiene en cuenta que, aunque con nuestra frecuencia de reloj podríamos conseguir trabajar con mayores resoluciones, mantener la frecuencia de refresco de 60Hz y el color real exige reducirla ligeramente.

Por todo lo anterior, nuestra zona de video visible es de 628 píxeles por 480 líneas, con una frecuencia de refresco vertical de 60 Hz (60 imágenes por segundo) y con color real con formato 5:5:5.

En lo relativo a las **memorias**, disponemos de dos bancos de memoria Ram de 512k posiciones de 16 bits cada una, con búfer de datos simple de 16 bits, con lo que nuestras comunicaciones con éstas van a ser con un ancho de palabra de 2 bytes.

Los tiempos de acceso a la memoria son diferentes para **lecturas y escrituras**. Las lecturas no presentan especiales problemas. Las escrituras, además de requerir un tiempo mayor, plantean el problema inherente a toda escritura: la necesidad de mantener la coherencia en los tiempos de direcciones y datos. La solución dada pasa por la introducción de un interfaz específico para emular un búfer doble, con el fin de sincronizar el acceso por parte de los componentes e independizar las escrituras de las lecturas.

Una restricción añadida es la **compartición de líneas**. En el banco derecho, las ocho líneas menos significativas del bus de datos son las mismas que se utilizan en la comunicación con el RamDac, por lo que no es posible realizar en un mismo ciclo la escritura en memoria en este banco y el refresco mediante el RamDac. Esto impide que podamos emplear el banco derecho para el procesado de las imágenes.

La frecuencia y cantidad de datos en la señal de entrada de video, junto con todas las consideraciones ya expuestas, obliga a emplear un patrón de alternancia estricta entre ciclos de lectura y escritura, utilizando además todos los ciclos de reloj. A su vez, el tiempo de ciclo está muy ajustado entre todos los componentes, lo que lleva a realizar muchos de los cálculos (señales de control, direcciones de acceso, ...) previamente al momento en que se necesitan.

Finalmente, queremos destacar que uno de los principios de diseño que más hemos ponderado ha sido una **descomposición modular** que favorezca los cambios y la reutilización, frente a una especificidad que llevaría a un diseño algo más optimizado.

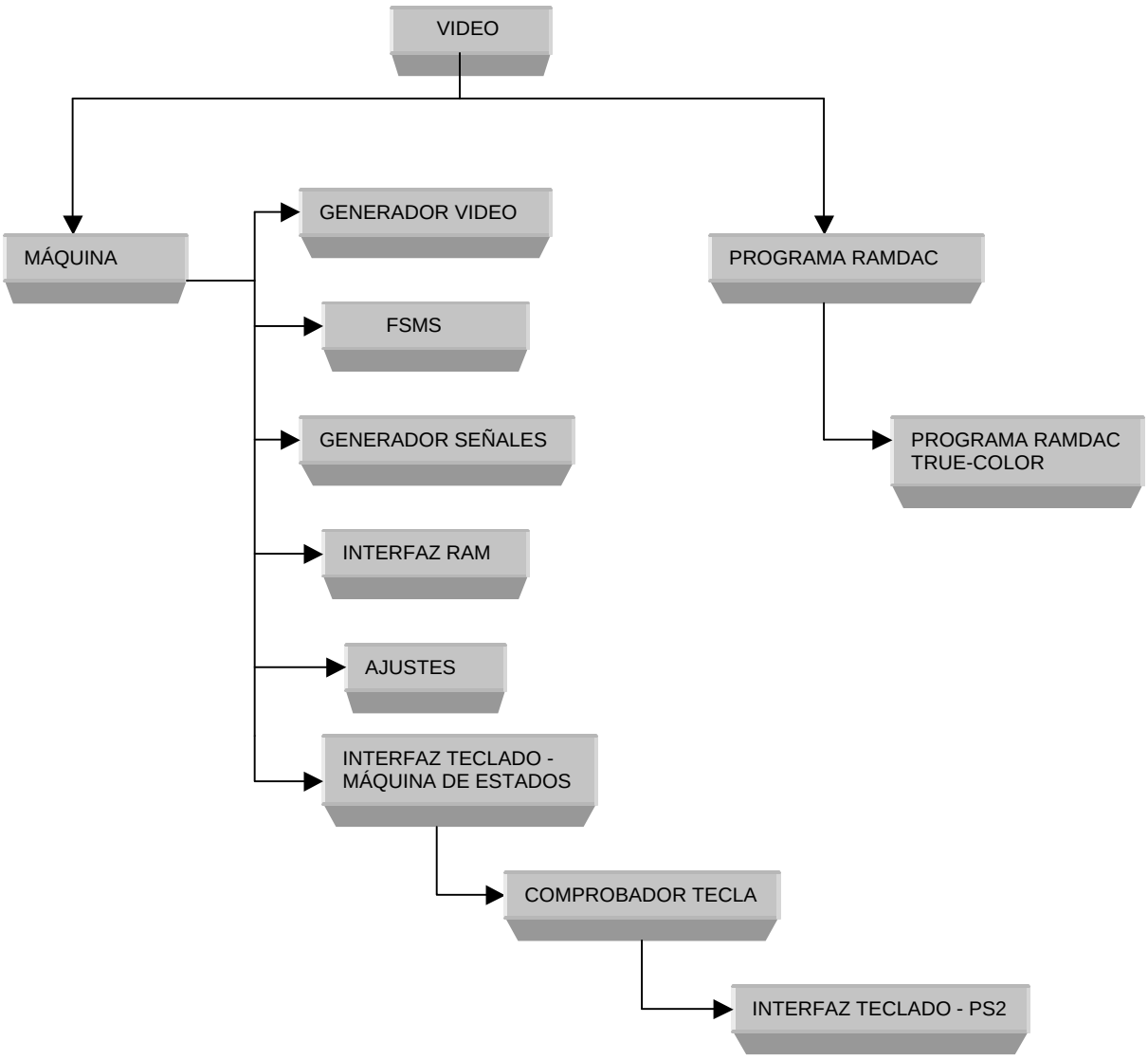
Las secciones siguientes desarrollan estos contenidos con mayor detalle.

3.4. Descomposición Modular

3.4.1. Descripción del Sistema

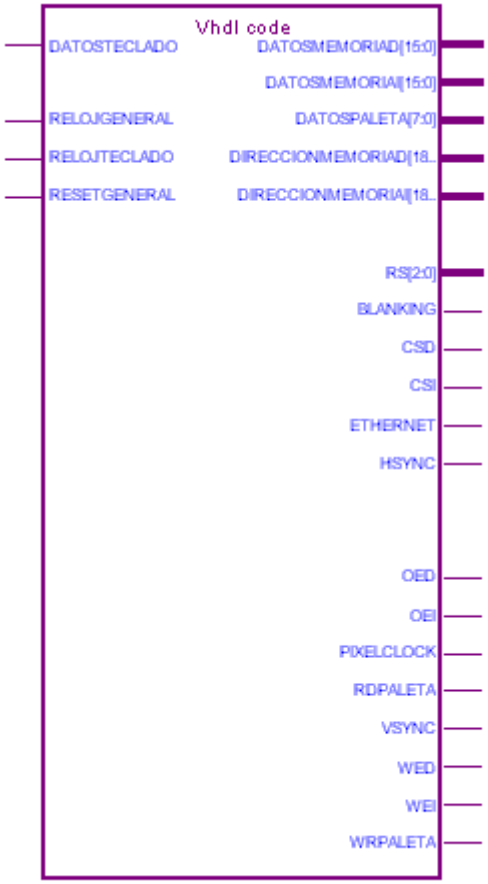
El sistema está dividido en una serie de módulos donde cada uno está implementado en un archivo "**.vhd*" diferente.

La jerarquía de módulos es la siguiente:



3.4.2. Vídeo

Es el módulo superior de la jerarquía. Contiene una instancia del módulo *máquina* y una del módulo *programaRamDac*. Se encarga de la inicialización (se empieza a programar el RamDac). Una vez que se ha inicializado, espera a que el RamDac esté programado. Cuando se acaba de programar el ramdac se pasa a un estado de funcionamiento donde el que gestiona todo el sistema es el módulo máquina.



Puertos agrupados por funcionalidad

RelojGeneral(in): es el reloj de la placa. Su frecuencia es de 50 MHz.
ResetGeneral(in): reset del sistema. Es asíncrono y activo a baja. Conectado a un switch de la placa

Teclado
DatosTeclado (in): son los datos que se leen del teclado. Es una línea bidireccional que sólo usamos en modo lectura. Es de un único bit porque el teclado transmite los datos en serie (cada tecla son 8 bits). Datos teclado se transmite a interfaz teclado-máquina de estados, y éste a su vez a los módulos que tiene por debajo(comprobador de tecla, interfaz teclado-PS/2).
RelojTeclado(in): reloj del teclado.

Señales para la programación del RamDac
WrPaleta (out): señal de escritura en la paleta (activa a baja). No debe estar activa a la vez que la señal de lectura (RDPaleta)
RdPaleta(out): señal de lectura de la paleta (activa a baja).
RS(2..0)(inout): líneas de selección de registro (register select). Se selecciona si se accede al registro de direcciones, a la RAM donde se almacena la paleta de colores, a los registros de overlay o se lee la máscara de registro. Tanto en la lectura como en la escritura RS se latchea en el flanco de bajada de WRPaleta o RDPaleta.
datosPaleta(7..0)(inout): línea bidireccional de datos con el RamDac. En la escritura los datos se latchean en el flanco de subida de WRPaleta.

Señales para la sincronización del Ramdac
pixelClock(out): reloj del RamDac (50 MHz)
hSync(out): sincronización horizontal (activa a baja)
vSync(out): sincronización vertical (activa a baja)
blanking(out): señal de blanking (activa a baja). Cuando la señal está activa se ignora el pixel recibido.

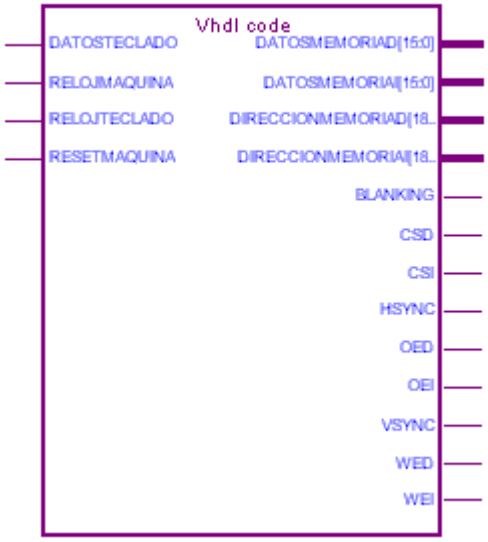
Banco izquierdo SRAM
WEI(out): write enable de la SRAM (activa a baja). Esta señal tiene prioridad sobre Output Enable, es decir, si está a 0 se va a escribir en el banco derecho independientemente del valor de la señal OEI.
OEI(out): output enable de la SRAM (activa a baja). Para realizar una lectura esta señal tiene que estar a 0 y WEI a 1.
CSI(out): chip enable de la SRAM (activa a baja). Para que se puedan realizar lecturas o escrituras tiene que valer 0.
direccionMemoriaI(18..0)(out): dirección de la memoria
datosMemoriaI(15..0) (inout): línea bidireccional de datos con la SRAM. A través de este buffer se escriben y se leen los datos del banco izquierdo.

Banco derecho SRAM
WED(out): write Enable de la SRAM (activa a baja). Esta señal tiene prioridad sobre Output Enable, es decir, si está a 0 se va a escribir en el banco derecho independientemente del valor de la señal OED.
OED(out): output enable de la SRAM (activa a baja). Para realizar una lectura esta señal tiene que estar a 0 y WEI a 1.
CSD(out): chip enable de la SRAM (activa a baja). Para que se puedan realizar lecturas o escrituras tiene que valer 0.
direccionMemoriaD(18..0) (out): líneas de dirección con la SRAM derecha.
datosMemoriaD(15..0) (inout): línea bidireccional de datosde datos con la SRAM derecha. En este buffer se escriben y se leen los datos del banco derecho.

Otras señales
ethernet: capacitación de Ethernet (activo a baja). Esta señal se pone fijo a uno para desactivarla

3.4.3. Máquina

El módulo *máquina* se encarga de generar los datos de escritura y las direcciones de lectura y escritura en memoria ,así como las señales para realizar dichas lecturas y escrituras. Genera también las señales de *blanking* y sincronismo horizontal y vertical .



Puertos agrupados por funcionalidad

resetMaquina(in): señal de inicialización (activa a baja)
relojMaquina(in): reloj de entrada (50 MHz)

Teclado
relojTeclado(in): reloj del teclado
datosTeclado(in): datos del teclado

Señales para la sincronización del RamDac
hSync(out): sincronización horizontal (activa a baja)
vSync(out): sincronización vertical (activa a baja)
blanking(out): señal de blanking (activa a baja)

Banco izquierdo SRAM
WEI(out): Write Enable de la SRAM (activa a baja)
OEI(out): Output Enable de la SRAM (activa a baja)
CSI(out): Chip Enable de la SRAM (activa a baja)
direccionMemoriaI(18.0) (out): dirección de la memoria
datosMemoriaI(15..0) (inout): búfer de datos con la SRAM izquierda

Banco derecho SRAM
WED(out): Write Enable de la SRAM (activa a baja)
OED(out): Output Enable de la SRAM (activa a baja)
CSD(out): Chip Enable de la SRAM (activa a baja)
direccionMemoriaD(18..0) (out): dirección de la memoria del banco derecho
datosMemoriaD(15..0) (inout): búfer de datos con la SRAM derecha

3.4.4. Generador de Vídeo

Este módulo simula un Codec de vídeo. Genera muestras de 16 bits con formato RGB 5:5:5 a una frecuencia de 25 MHz, además de la direcciones de memoria en la que se tiene que escribir cada píxel. Se ha incluido la generación de las direcciones de escritura para simplificar el módulo máquina. De no haberse hecho así, el módulo máquina tendría que contar los pixels que va recibiendo y generar las direcciones de escritura.

Puertos de entrada:

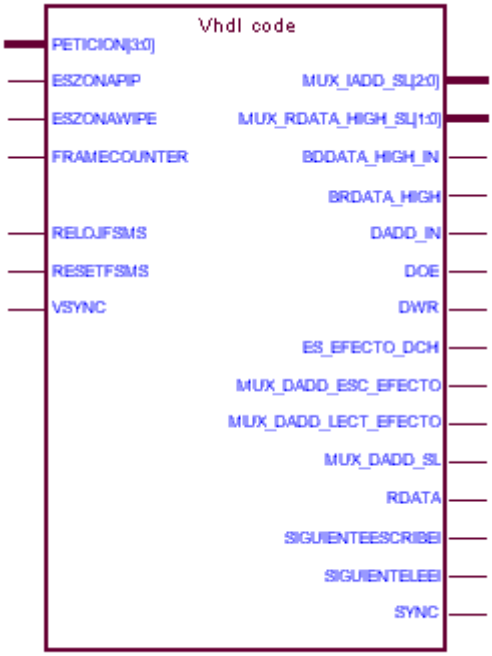
relojVideo(in): entrada del reloj a 50MHz (funciona con flanco de subida).
resetVideo(in): señal de reset asíncrona(activa a baja).

Puertos de salida:

datosVideo(15..0)(out): valor RGB 5:5:5 del pixel generado.
direccionVideo(18..0)(out): dirección del pixel generado.

3.4.5. FSMS

Módulo que implementa la máquina de estados. Tenemos dos máquinas:la máquina de ciclos y la máquina de efectos. La máquina de ciclos controla si lo que tenemos que hacer una lectura o una escritura, en que banco de memoria escribimos y de qué banco de memoria leemos. Esta alternancia entre los diferentes estados dependen de la máquina de efectos que controla qué efecto se ha seleccionado.



Puertos de entrada:

frameCounter: señal del contador de frames .
esZonaPip: señal que indica si estamos en zona pip(activo a alta)
esZonaWipe: señal que indica si estamos en zona wipe(activo a alta)
peticion(3..0): código del efecto (zoom, mosaico, pip..).
relojFsms: reloj de la máquina de estados(50 MHz).
resetFsms: reset asíncrono(activo a baja).
vSync: señal de sincronismo vertical.

Puertos de salida:

es_efecto_dch: señal de carga del registro es_efecto_dch.
bddata_high_in: señal de carga del buffer bddata_high_in(activo a alta).
brdata_high: señal de carga del buffer del brdata_high(activo a alta).
dadd_in: señal de carga del registro dadd_in.
doe: Output Enable de la SRAM (banco derecho).
dwr: Write Enable de la SRAM(banco derecho).
mux_dadd_lect_efecto: selección del multiplexor mux_dadd_lect_efecto.
mux_dadd_esc_efecto: selección del multiplexor mux_dadd_esc_efecto.
mux_dadd_sl: señal de selección del multiplexor mux_dadd_sl.
mux_iadd_sl(2..0): señal de selección del multiplexor mux_iadd_sl.

mux_rdata_high_sl: señal de selección del multiplexor mux_rdata_high_sl.
rdata: señal de carga del registro rdata.
sync: señal de sincronismo.

3.4.6. Interfaz RAM

Este módulo se encarga de gestionar el acceso al banco izquierdo de la SRAM. Permite el acceso a una SRAM de buffer simple simulando un doble buffer tanto para las direcciones y los datos. De esta forma se consigue realizar lecturas y escrituras en un único ciclo.Las señales de hazLectura y hazEscritura provienen de la fsms, y le indican la operación a realizar. El dato a escribir y la dirección de escritura vienen del generador de vídeo y la dirección de lectura de la máquina. El módulo a partir de estas entradas obtiene los datos y la dirección de memoria y las señales de CE,WE y OE.

Puertos de entrada:

relojInterfaz: reloj de entrada
resetInterfaz: reset del circuito (activo a baja)
hazLectura: señal para solicitar una lectura (activa a alta)
hazEscritura: señal para solicitar una escritura (activa a alta)
direccionDeLectura(18..0): dirección de la que queremos leer
direccionDeEscritura(18..0): dirección sobre la que queremos escribir
datoAEscribir(15..0): dato que queremos escribir en la memoria

Puertos de salida:

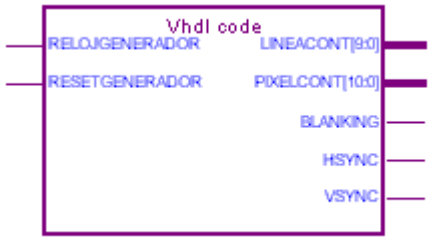
datoLeido(15..0):dato que hemos leido de la memoria
direccionMemoria(18 downto 0): bus de direccicones a la SRAM (banco izquierdo)
chipEnable: Chip Enable de la memoria (activo a baja)
outputEnable: Output Enable de la memoria (activo a baja)
writeEnable: Write Enable de la memoria (activo a baja)

Puertos de entrada/salida:

datosMemoria(15..0): bus de datos a la memoria(banco izquierdo).

3.4.7. Generador de Señales

Este módulo genera las señales necesarias para el controlador. Tiene como entradas el reloj a 50Mhz y el reset asíncrono y como salidas da las señales de *blanking*, sincronismo horizontal y vertical , contador de píxeles y contador de líneas. El contador de píxels se incrementa cada ciclo de reloj y el contador de líneas cuando hay una señal de hsync. A partir del contador de píxeles y del de líneas se obtiene la dirección de refresco base.



Puertos de entrada:

relojGenerador: reloj de entrada (funciona con flancos de subida,50 MHz)
resetGenerador: reset asíncrono (activo a baja)

Puertos de salida:
pixelCont(10..0): contador de píxeles
lineaCont(9..0): contador de lineas

Señales para la sincronización del RamDac
hSync: sincronización horizontal (de línea) (activa a baja)
vSync: sincronización vertical (de frame) (activa a baja)
blanking: señal de blanking (activa a baja) (1:zona visible;0:fuera)

3.4.8. Ajustes

Este módulo se encarga de calcular las direcciones de los diferentes efectos. Recibe como entradas la dirección de refresco base (obtenida a partir de los contadores de píxeles y líneas del generador de señales), la dirección de refresco registro (es la dirección de refresco pero latcheada), *tPetición* (código del efecto en el que estamos), *tAjuste* (código que indica si se ha pulsado un + o -) , el reloj de 50MHz y el reset asíncrono. Calcula las direcciones para los siguientes efectos: reflejo, mosaico, pip, rotación, strobe cíclico. Se encarga de aumentar y disminuir el tamaño del zoom y del mosaico, de cambiar la zona de pip, aumentar y disminuir la parte dinámica del wipe y aumentar y disminuir la velocidad del estrobe.

Puertos de entrada:

relojAjustes: reloj de entrada(50 MHz)
vsync: sincronismo vertical.
resetAjustes: reset asíncrono(activo a baja)
tAjuste(1..0) : codificación de las teclas +/-.
tPeticion (3..0) : codificación del efecto.
dirRefrescoBase(18..0): dirección obtenida a partir del generador de señales.
dirRefrescoReg(18..0) : dirección que va a utilizar la memoria (puede ser de zoom, refresco base, mosaico ...).

Puertos de salida:

esZonaWipe: señal que indica si estamos refrescando en zona wipe.
esZonaPip: señal que indica si estamos refrescando en zona pip.
frameCounter:señal de fin de cuenta de frame counter.
dirPip(18 ..0): dirección de refresco cuando el efecto es pip.
dirZoom(18 ..0): dirección de refresco cuando el efecto es zoom.
dirMosaico(18 ..0): dirección de refresco cuando el efecto es mosaico.
dirStrobeC(18 ..0): dirección de refresco cuando el efecto es strobe cíclico
dirFlip(18 ..0): dirección de refresco cuando el efecto es flip.
dirRotacion(18 ..0): dirección de refresco cuando el efecto es rotación.

3.4.9. Interfaz Teclado - Máquina de Estados

Este módulo gestiona las peticiones de teclado. Tiene como entradas el reloj del circuito, el reset asíncrono activo a baja, el reloj del teclado, datos del teclado (*PS2Data*) y la señal de sincronismo vertical. Genera un código de 4 bits para el efecto y un código de 2 bits para + y - .

Puertos de entrada:

clock: reloj de entrada(50Mhz)
reset: señal de reset asíncrona(activo a baja)
ps2Clk: reloj del teclado
ps2Data: datos del teclado
vsync: sincronismo vertical.

Puertos de salida:

registroestado(3..0): registro para almacenar el estado
registroajustes(1..0): registro para almacenar si se ha pulsado '+' o' -'

3.4.10. Comprobador Tecla

El módulo comprobador tecla es un eliminador de rebotes del teclado.Dadas las señales de reset, reloj reloj del teclado (*ps2clk*) y un dato del teclado (*ps2data*) devuelve si ha habido pulsación de una tecla, y el código de la tecla pulsada (8 bits).

Puertos de entrada:

clock: reloj de entrada(50Mhz)
reset: señal de reset asíncrona(activo a baja)
ps2Clk: reloj del teclado
ps2Data: datos del teclado

Puertos de salida:

tecla(7..0): código de la tecla pulsada
pulsacion: señal que indica si se ha pulsado una tecla (activo a alta)

3.4.11. Interfaz Teclado - PS/2

El módulo *interfaz teclado-PS2* recibe un reloj y un reset general, el reloj del teclado(*ps2clk*), los datos de entrada del teclado y devuelve el código de la tecla que ha sido pulsada y una señal que indica que ha recibido un nuevo dato del teclado.

Los datos del teclado se reciben en serie .

Puertos de entrada:

clock: reloj de entrada(50Mhz)
reset: señal de reset asíncrona(activo a baja)
ps2Clk: reloj del teclado
ps2Data: datos del teclado
newDataAck: espera a que le llegue un uno como confirmación.

Puertos de salida:

data(7..0): datos de salida del controlador.
newData: señal que se pone a uno cuando se manda un dato y se pone a esperar confirmación.

3.4.12. Programa RamDac

Este módulo contiene un componente *programaRamdacTrueColor*..

Recibe el reloj de 50Mhz, un reset asíncrono y la señal de inicio de programación que pasa al componente *programaRamdacTrueColor* .Permanece en espera hasta que se ha finalizado la programación del ramdac. Una vez finalizada la programación pasa a un estado de espera y activa una señal de fin de programación.Este módulo se ha implementado para facilitar el paso de programación en TrueColor a programación en grises.

Puertos de entrada:

relojProgramador: reloj de entrada (a 50MHz)
resetProgramador: reset del circuito (activo a baja)
inicioProgramacion: señal de inicio (activa a alta)

Puertos de salida:

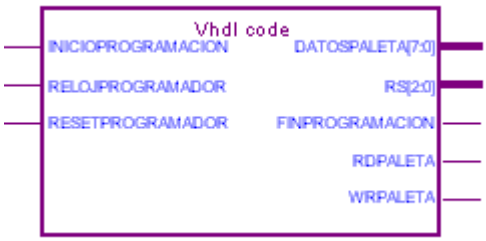
finProgramacion: señal de finalización (activa a alta)
WrPaleta: señal de escritura en la paleta (activa a baja)
RdPaleta: señal de lectura de la paleta (activa a baja)

Puertos de entrada/salida:

RS(2..0): líneas de selección de registro (register select)
datosPaleta(7..0): línea bidireccional de datosPaleta con el RamDac

3.4.13. Programa RamDac True Color

El módulo programa el RamDac para que funcione con color real en el formato 5:5:5 funcionando con un solo flanco (single-edge). Recibe como entradas el reloj a 50 MHz, el reset asíncrono (activo a baja) e inicioProgramación que se pone en alta para comenzar a programar. La señal *finProgramación* indica que se ha terminado de programar.



Puertos de entrada:

relojProgramador: entrada del reloj a 50MHz
resetProgramador: reset del circuito (activo a baja)
inicioProgramacion: para que comience a programar (activo a alta)

Puertos de salida:

finProgramacion: indica que ha terminado la programación (activo a alta)
WrPaleta: señal de write de la paleta del RamDac (activa a baja)
RdPaleta: señal de lectura de la paleta del RamDac (activa a baja)

Puertos de entrada/salida:

RS(2..0): selección de registro de la paleta (Register Select)
datosPaleta(7..0): línea bidireccional con la paleta del RamDac

3.5. Diseño del Controlador y Ruta de Datos

3.5.1. Disposición de Memoria

En el modo de funcionamiento de color real se emplean 16 bits para la codificación de cada píxel. Dado el tamaño de nuestra zona visible, los requisitos totales de memoria para una imagen son 628x480x16 bits. Por lo tanto, se puede almacenar una imagen completa en cada uno de los bancos de memoria (2^{19} palabras). En el mapa de memoria se colocan contiguos todos los píxeles de una misma línea horizontal, separada cada una de ellas por el tamaño correspondiente a la zona no visible de sincronización entre líneas.

En cuanto a las escrituras en memoria, el banco izquierdo se emplea para almacenar directamente la imagen recogida por la entrada de video. El banco derecho se emplea para imágenes ya procesadas de acuerdo con cada efecto. La escritura de la nueva imagen entrante en el banco izquierdo se mantiene siempre.

El refresco de datos se efectúa desde ambos bancos. En el modo de reproducción normal, las lecturas y escrituras se hacen exclusivamente en el banco izquierdo, de forma alterna. Para los efectos puede que se empleen ambos bancos, según se describa en cada implementación particular.

El flujo de datos básico consiste en trasladar la imagen del banco izquierdo al RamDac. Se requieren dos condiciones de sincronismo: del píxel que se lee de memoria con el que corresponde refrescar al RamDac; y de la parte alta y baja de la palabra que representa el píxel con la media palabra que el RamDac espera muestrear.

La operación de escritura del banco derecho debe encajarse con el muestreo del RamDac, ya que sus líneas de entrada están compartidas con las líneas de datos menos significativas del banco de memoria derecho. La misma estructura proveerá la segmentación del píxel, que se recibe de la memoria izquierda en 16 bits cada 25MHz, y tiene que entregarse al RamDac en 8 bits cada 50MHz.

3.5.2. Idea del Controlador

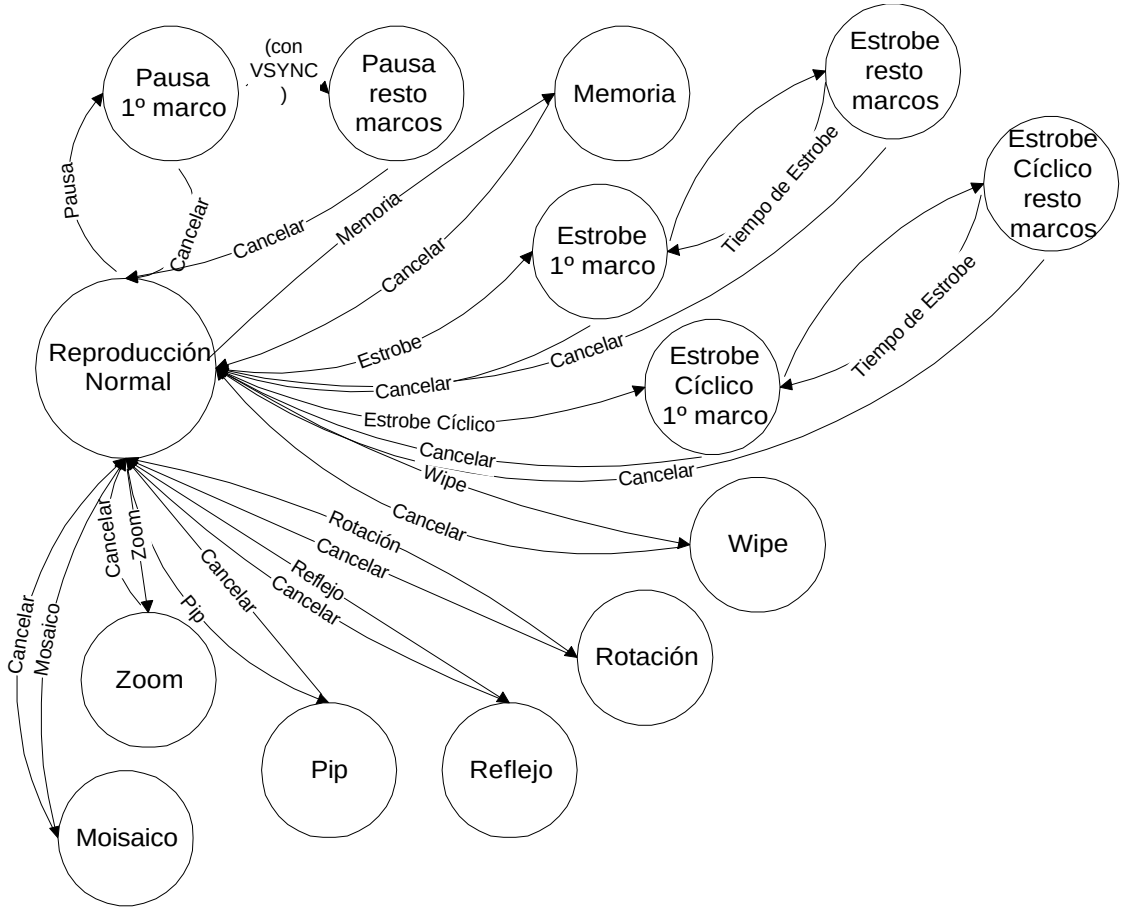
Las operaciones básicas de la ruta son cuatro. A saber: escritura en el banco izquierdo, y refresco; escritura en el bando derecho, y refresco. Corresponden con el tipo de operación que se puede realizar en un ciclo de 50MHz, y determinan las señales de control de la ruta para el ciclo.

Estado	Descripción
eI	escribe en el banco izquierdo
rI	refresca del banco izquierdo
eD	refresca del banco izquierdo y escribe sincronizadamente en el derecho
rD	refresca del banco derecho

Los nombres de los estados se refieren al uso que se hace de las líneas de memoria en ese estado. Tanto **eD** como **rI** refrescan del banco izquierdo, la diferencia es que **rI** preserva el contenido de la memoria derecha, mientras que **eD** lo actualiza.

El control se divide en una jerarquía de dos niveles. Los cuatro tipos de operaciones básicas están representados en los estados de la máquina **Ciclos**. El tipo de alternancia entre estos cuatro estados, junto con las direcciones y datos que se usan para atacar a la memoria, dependen del efecto que está en curso. Los efectos se representan como estados de la máquina **Efectos**.

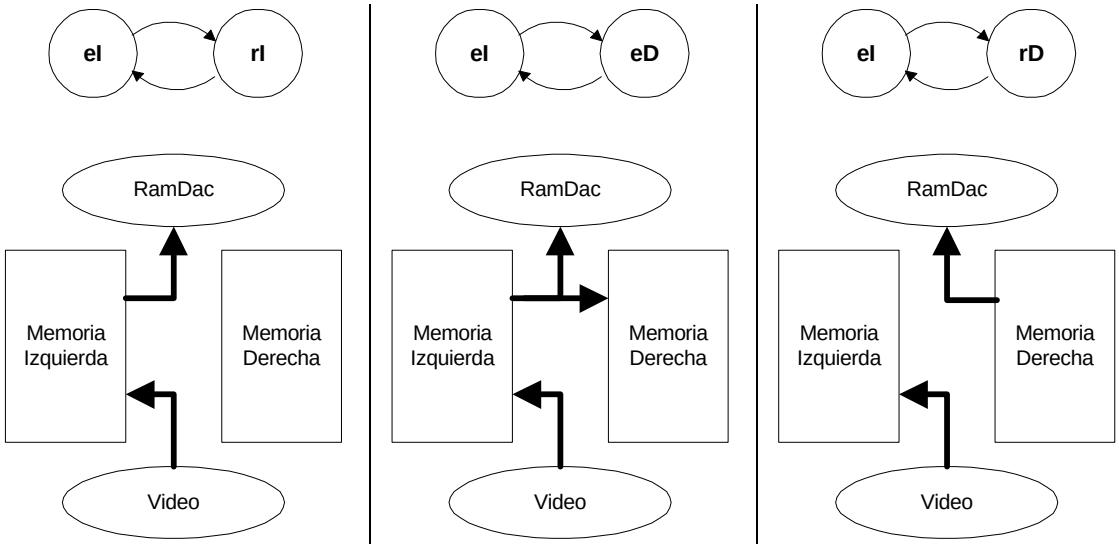
El estado de **Efectos** es una entrada a la máquina **Ciclos**. El efecto deseado puede cambiar de una imagen mostrada a la siguiente, pero no de un píxel al consecutivo (el reloj de la máquina es la señal de sincronismo vSync).



Efecto	Alternancia de Ciclos que lo componen
Pausa	Durante el primer marco del efecto se alterna entre los estados **eI** y **eD**. Desde el segundo marco se alterna entre los estados **eI** y **rD**
Memoria	Se alterna entre **eI** y **rD**.
Estrobe y Estrobe Cíclico	En el primer marco, se alterna entre **eI** y **eD**, para refrescar mientras se escribe en la derecha. En el resto de marcos se alterna entre **eI** y **rD**, para refrescar la imagen congelada y escribir la entrante La diferencia entre ellos está en las direcciones usadas para el escalado.

Wipe	Alternancia normal entre **rI** y **eI** en la zona de no efecto y entre **rD** y **eI** en la zona de efecto.
Pip	Alternancia normal entre **rI** y **eI** en la zona de no efecto y entre **rD** y **eI** en la zona de efecto.
Zoom	Alternancia normal entre **rI** y **eI**.
Mosaico	Alternancia normal entre **rI** y **eI**.
Reflejo	Alternancia normal entre **rI** y **eI**.
Rotación	Alternancia normal entre **rI** y **eI**.

El flujo de datos de video según el modo de alternancia entre estados tiene, por tanto, las posibilidades indicas en este diagrama:



Reglas del cambio de estado en la máquina de **Ciclos**. El único estado donde hay bifurcaciones es eI; dependen del efecto en curso y, en algunos de ellos, de si la zona que se refresca está afectada o no por el efecto.

Ciclo	efecto en vigor	zona de efecto	Ciclo Siguiente
eI	ninguno	-	rI
	pausa 1er marco	-	eD
	pausa resto marcos	-	rD
	memoria	-	rD
	estrobe 1er marco	-	eD
	estrobe resto marcos	-	rD
	zoom	-	rI
	moisaico	-	rI
	pip	0	rI
		1	rD
	wipe	0	rI
		1	rD
rI			eI
rD			eI
Ed			eI

3.5.3. Flujo Básico en la Ruta de Datos

El funcionamiento general es como sigue. Los contadores de refresco indican la dirección que corresponde al píxel que se va a refrescar seguidamente. En el modo normal se usa esta dirección para leer de la memoria izquierda el valor correspondiente, pasándolo a su interfaz a través del multiplexor MUX_IADD_IN. Si cualquiera de los efectos de Zoom, Mosaico, Reflejo o Rotación está activo, se selecciona la dirección adecuada, que se ha calculado en el módulo de ajustes a partir de la dirección de refresco.

El dato que se extrae de la interfaz con la memoria izquierda se almacena en el registro RDATA. Los 16 bits obtenidos en paralelo tienen que pasarse en serie al RamDac. En el registro RDATA_HIGH se carga la media palabra correspondiente, mientras la siguiente operación de lectura prosigue por la parte adecuada de la ruta.

Esta misma operación de refresco se aprovecha en determinados efectos para hacer la escritura en la memoria derecha, ya que las líneas menos significativas están compartidas con el RamDac. Desde el multiplexor MUX_DADD se selecciona la dirección de escritura según el efecto (escritura completa de una imagen en memoria, cogiendo la dirección del registro IADD_IN, o bien la dirección de Estrobe).

En el refresco desde el banco derecho es necesario introducir los búferes triestado BRDATA_HIGH y BDDATA_HIGH_IN, para poner a alta impedancia la salida de los registros mientras se lee de las líneas de memoria.

El bloque TRIPLE_SYNC se encarga de retener (mediante el llenado de un *pipeline*) los valores de sincronismo que corresponden al píxel, desde el momento en que se calculan en los contadores de refresco, hasta el momento que el RamDac muestrea ese píxel.

En el análisis del valor de las señales de control de la ruta, hay que tener en cuenta que en todos los estados se realizan acciones preparatorias para la acción correspondiente al ciclo siguiente. Así, por ejemplo, las señales que preparan un próximo ciclo de escritura tienen valores comunes en **rI**, **eD** y **rD**.

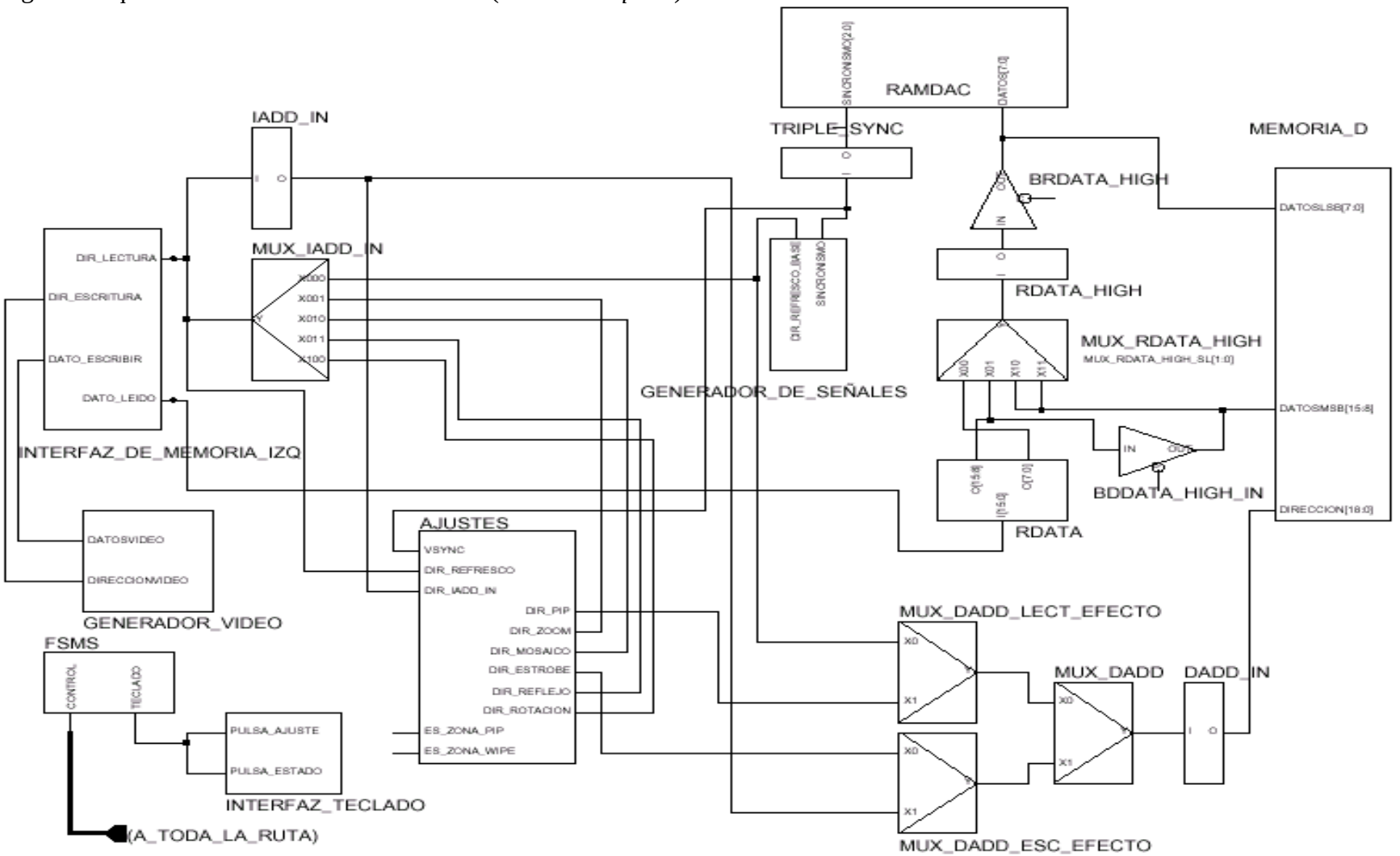
Señales de control de la ruta de datos según el estado actual de ciclos y el estado siguiente:

	Señales de control	eI	rI	eD	rD
Banco Izquierdo	SIGUIENTELEEI	0 si a rD 1 si a eD 1 si a rI	0	0	0
	SIGUIENTEESCRIBEI	0	1	1	1
	MUX_IADD_SL	000 si normal 001 si Zoom 010 si Mosaico 011 si Reflejo 100 si Rotación	000	000	000
Selección de Ciclo	ES_EFECTO_DCH	0	1	0	0
Banco Derecho	DADD_IN	0	1	1	1
	MUX_DADD_SL	0 (d)	0 (d)	1	0
	MUX_DADD_LECT_EFECTO	0 (d)	0 (d)	0 (d)	1 si Pip 0 sino
	MUX_DADD_ESC_EFECTO	0 (d)	0 (d)	0 si Estrobe Cíclico 1 sino	0
	BDDATA_HIGH_IN	0	0	1	0
	MUX_RDATA_HIGH_SL	00	01	01	10
	BRDATA_HIGH	1	1	1	0
	DWR	1	1	0	1
	DOE	1	1	1	0
Salida al RamDac	RDATA	0	1	1	0
	SYNC	0	1	1	1

(d) representa *don't care* lógicos, que han sido explicitados en la codificación.

El módulo "*fsms.vhd*" implementa el controlador.

Diagrama esquemático de la ruta de datos básica (módulo *máquina*):



Interfaz de memoria con el banco izquierdo

El diseño que hemos implementado presenta una asimetría: el banco derecho de memoria se gestiona directamente desde el controlador, mediante las salidas reseñadas de la máquina de ciclos; por el contrario, el banco izquierdo se gestiona mediante un interfaz interpuesto entre la memoria y el controlador.

La gestión de la memoria izquierda se ha independizado porque son dos entidades distintas las que se comunican con ella. Las escrituras, con el dato y dirección, vienen directamente de la entrada continua de video. Así únicamente tiene que conocer a la interfaz y el diseño es menos acoplado. Las peticiones de lectura se emiten desde el controlador, con la dirección que corresponda según el efecto en curso.

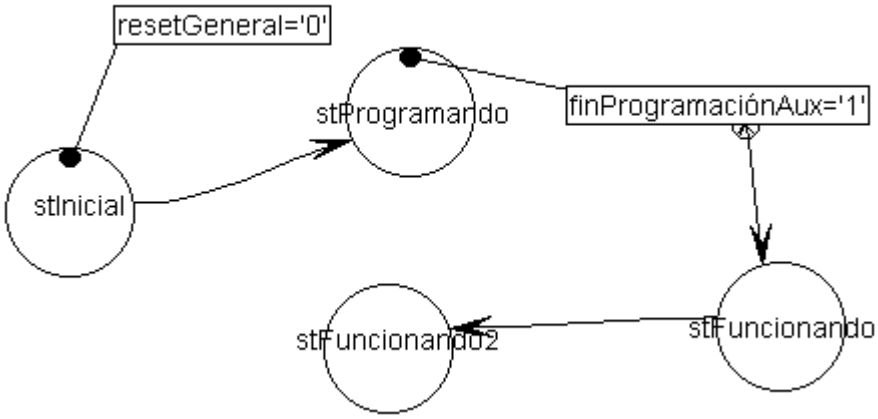
La interfaz de memoria se encarga de la armonización de los dos tipos de peticiones. Los datos de entrada se muestrean y almacenan internamente hasta que se efectúa la escritura, por tanto la dependencia de otros módulos es menor. Por otra parte, los datos leídos también se muestrean y almacenan en un **búfer de salida**, en cuanto se recibe la petición siguiente. Así se puede asegurar que la entrada a la ruta de datos permanece estable durante todo el tiempo necesario, mientras que se liberan las líneas de datos de la memoria para realizar la siguiente escritura. Ésta es la principal limitación que se supera con la interfaz.

La gestión de la memoria derecha se mantiene directamente en el controlador por dos motivos. Primero, la alternancia entre escrituras y lecturas no se da en ciclos consecutivos sino en imágenes consecutivas. Esto elimina la necesidad de un doble búfer. Segundo, la compartición de las líneas de datos menos significativas del banco derecho con la entrada de datos del RamDac es una complicación añadida que hace conveniente un control unificado.

3.5.4. Vídeo

Como puede verse en el diagrama de módulos el tiene dos componentes : *programaRamDac* y *Máquina*. Como se puede ver en la figura, pasa por tres estados:

- stInicial: en este estado resetea los módulos Máquina y programaRamDac.Se llega a este estado cuando hay un reset.
- stProgramando: en este estado programa el RamDac para que funcione en true color 5:5:5. Permanece en este estado hasta que el RamDac esté programado.
- stFuncionando: se llega a este estado una vez que el RamDac ya se ha programado.
- stFuncionando2: estado en el que permanece hasta que se vuelva a hacer un reset del sistema



3.5.5. Máquina

Este módulo gestiona y controla todo el flujo de datos del circuito en función de los datos suministrados, las señales internas y las peticiones de los usuarios. Instancia los siguientes módulos:

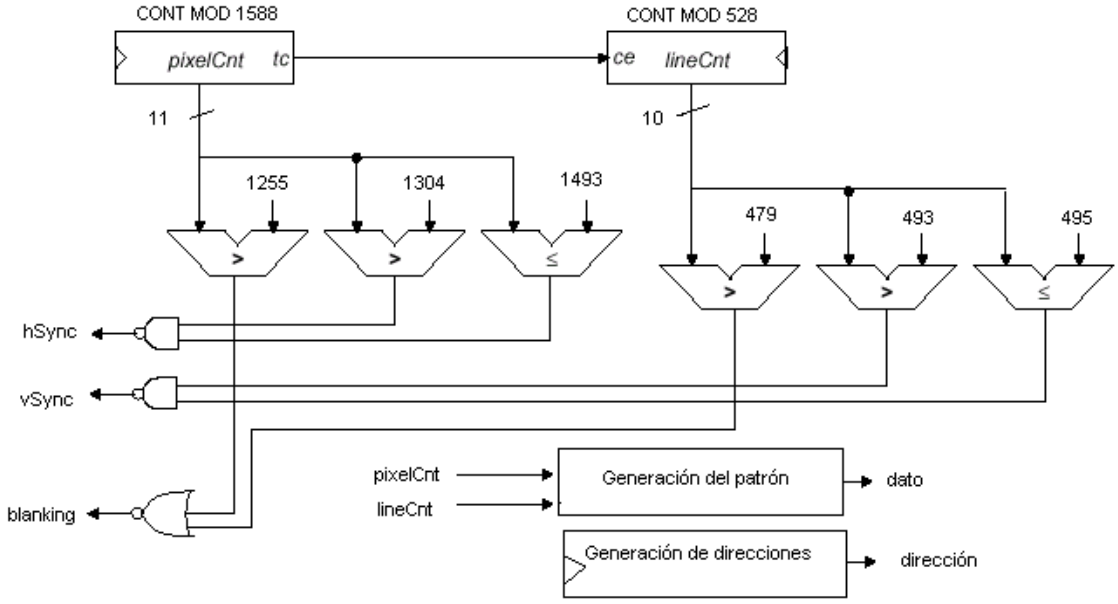
- Generador de vídeo
- Generador de señales
- FSMS
- Ajustes
- Interfaz RAM
- Interfaz teclado - máquina de estados

Para la interconexión de estos componentes y la comunicación con el bloque derecho de memoria se incluyen algunos componentes sencillos (registros, búferes, multiplexores...). Para una información más detallada consultar la sección donde se explica la ruta de datos y el código del módulo, maquina.vhd, que se encuentra en el apéndice.

3.5.6. Generador de Vídeo

Este módulo simula el comportamiento de un codec de vídeo. Genera muestras de 16 bits con formato RGB 5:5:5, a una frecuencia de 25 MHz, además de generar la dirección de memoria correspondiente a ese píxel.

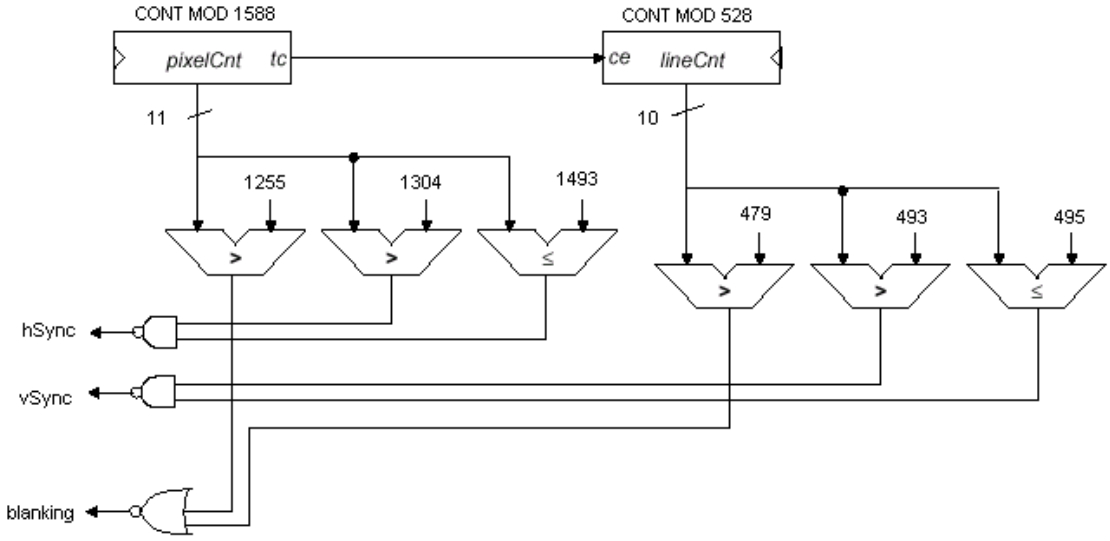
La implementación de este módulo se basa en un contador de píxeles, un contador de líneas y comparadores. El contador de píxeles se incrementa cada ciclo de reloj, y el de líneas con la señal de fin de cuenta del contador de píxeles. Los comparadores junto con algunas puertas se usan para obtener las señales de sincronismo horizontal, vertical y blanking. El patrón que se genera se obtiene a partir de los valores de los contadores. En la siguiente figura se puede ver el diseño del módulo:



3.5.7. Generador de Señales

Este módulo genera las señales necesarias para el controlador, que funciona a 50 MHz. Tiene como entradas un reloj de 50 MHz y un reset asíncrono, y como salidas nos da las señales de sincronismo (horizontal, vertical y blanking) , y los contadores de líneas y píxeles.

El circuito, como puede verse en la figura, es muy similar al del generador de vídeo.



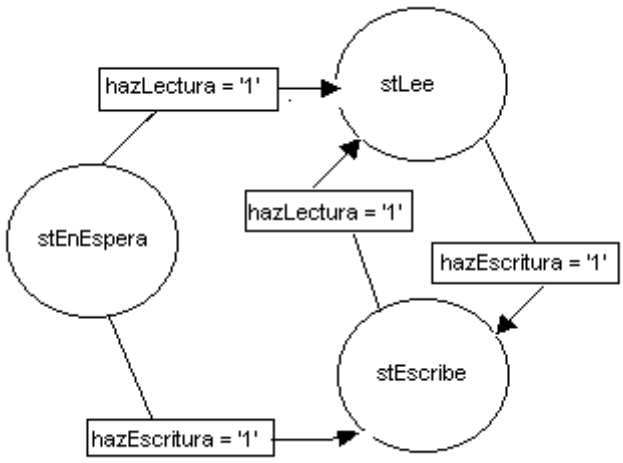
3.5.8. Interfaz RAM

Es un interfaz que nos permite acceder a una memoria RAM de bus único. Implementa los accesos a la memoria en un único ciclo, lo que nos permite accesos en ciclos consecutivos.

El objetivo principal de este módulo es mejorar la temporización en el acceso. Para ello se ha implementado una estructura de doble búfer, tanto para las direcciones como para los datos, de modo que disponemos de manera independiente de una conexión en sentido salida (lectura) y otro en sentido entrada (escritura), que se multiplexarán dentro del módulo de manera adecuada. Esto permite a los componentes que se conecten al módulo disponer de más tiempo. Tanto los datos como las direcciones se registran internamente, por lo que no es necesario que lo hagan los módulos externos que se conecten con el interfaz. Sin embargo, es muy conveniente que los que lo vayan a utilizar para lectura , registren el puerto de datos leídos para asegurar que cogen el dato que solicitaron. El proceso, tanto para lectura como escritura consiste en realizar una petición de operación poniendo la señal correspondiente a alta y los datos y dirección con los que queremos que se opere. Como la operación se hará efectiva en el siguiente ciclo y los registros no se cargarán hasta el flanco de reloj, el orden en que fijemos las señales es totalmente independiente, con lo que dotamos al interfaz de gran flexibilidad.

El controlador de este módulo tiene los siguientes estados:

- stEnEspera: se está en este estado cuando no hay ni petición de lectura ni petición de escritura
- stLee: se llega a este estado cuando ha habido una petición de lectura. Se sigue en este estado mientras haya peticiones de lectura. Si hay una petición de escritura se pasa al estado stEscribe y si no hay ninguna petición al estado stEnEspera.
- stEscribe: a este estado se llega con una petición de escritura. Al igual que en el estado stLee, se permanece en este estado mientras siga habiendo peticiones de escritura. Si hay una petición de lectura se pasa al estado stLee y si no hay peticiones al estado stEnEspera.



3.5.9. Ajustes

Este módulo implementa los componentes que gestionan las modificaciones que se puedan hacer sobre los efectos, así como generar las direcciones de acceso a memoria de los diferentes efectos.

A continuación, se explica de manera más detallada la implementación de los efectos.

3.5.9.1. Pausa

Con este efecto conseguimos **congelar la imagen que se está mostrando en ese momento en pantalla**.

Como el generador de video no se detiene en ninún momento, lo que hacemos realmente es capturar la primera imagen que se genere a partir de la solicitud del efecto, almacenarla en el banco derecho de la memoria y después, continuar visualizando desde este banco, para que podamos seguir capturando lo que se esté generando y almacenarlo en el banco izquierdo.

Hacerlo, en principio no tendría mucho sentido, y habría sido suficiente con que hubiésemos detenido la escritura en el banco izquierdo de la memoria y únicamente accediéramos a ella en modo lectura, con lo que estaríamos visualizando la última imagen que se hubiera almacenado. Sin embargo, queremos que sea posible recuperarla en cualquier momento que el usuario lo solicite (mediante el efecto memoria que veremos más adelante) por lo que si hacemos esto, al cancelar el efecto y volver al modo normal, estaríamos sobreescribiendo lo que hubiese en el banco izquierdo y no sería posible volver a recuperarlo.

Aún así, el efecto sigue siendo muy sencillo y lo único que tenemos que hacer es, tras una petición de pausa, pasar por un estado intermedio en el que, durante toda una pantalla (frame), estamos aprovechando los datos que estamos leyendo del banco izquierdo para, a la vez que se refrescan y se le envían al RamDac, se van almacenando en el banco derecho de la memoria. En cuanto hayamos hecho esto durante toda una pantalla, pasaremos al estado de pausa propiamente dicho, en el únicamente refrescaremos el banco derecho hasta que el usuario cancele el efecto.

Por cómo hemos diseñado nuestro sistema, las escrituras y las lecturas se hacen siempre a la vez en ambos bancos (en caso de que haya que hacerlas), por lo que es necesario que almacenemos en un registro la dirección que estamos usando para refrescar el dato del banco izquierdo, para poder utilizarla en el siguiente ciclo, en el caso de que sea necesario acceder al banco derecho para escribir.

3.5.9.2. Memoria

Mediante este efecto **recuperamos la imagen que previamente hemos almacenado en memoria**.

Normalmente ésta se habrá capturado mediante el efecto de pausa aunque, por cómo hemos implementado nuestro sistema, ésta también podrá provenir de la última imagen que se almacenara mediante un estrobe u estrobe cíclico.

Quizá sea el efecto más sencillo pues lo único que hay que hacer es que la máquina de estados, tras un ciclo de escritura en el banco izquierdo (que continua haciéndose siempre), en vez de pasar a un estado de refresco del banco izquierdo, pase a un estado de refresco del banco derecho, dando las señales oportunas que configuren la ruta de datos de manera conveniente.

En este caso, como la dirección que usamos es la misma que estaríamos usando en el modo normal para acceder al banco izquierdo, no necesitamos ningún hardware adicional sino que únicamente tenemos que conectar dicha dirección al multiplexor que le corresponde para que sea usada en el caso de que estemos en este efecto.

3.5.9.3. Mosaico

Con el mosaico lo que pretendemos es **disminuir gradualmente la resolución de las imágenes**, es decir, disminuir el número de píxeles que utilizamos para mostrarlas en pantalla.

Una manera sencilla de conseguirlo es aumentar el tamaño de píxel que utilizamos con lo que nos cabrán menos en la pantalla. La consecuencia inmediata de esto es que hay que desechar un número determinado de píxeles.

Hay varios algoritmos para conseguir este efecto, pero nosotros hemos optado por uno sencillo que no sobrecargue nuestro sistema y que sea fácilmente implementable en hardware.

Aunque las imágenes se siguen capturando con total normalidad, a la hora de mostrarlas en pantalla, lo que hacemos es refrescar varias veces el mismo píxel con lo que conseguimos simular ese aumento del tamaño. Para entenderlo, veamos un ejemplo para un factor de 2:

Imagen Original

	0	1	**2**	3	**4**	5	...
0	**0,0**	0,1	**0,2**	0,3	**0,4**	0,5	...
1	1,0	1,1	1,2	1,3	1,4	1,5	...
2	**2,0**	2,1	**2,2**	2,3	**2,4**	2,5	...
3	3,0	3,1	3,2	3,3	3,4	3,5	...
4	**4,0**	4,1	**4,2**	4,3	**4,4**	4,5	...
5	5,0	5,1	5,2	5,3	5,4	5,5	...
...	...	...	...	...	...	...	...

Imagen Mosaico

	0	1	**2**	3	**4**	5	...
0	**0,0**		**0,2**		**0,4**		...
1							...
2	**2,0**		**2,2**		**2,4**		...
3							...
4	**4,0**		**4,2**		**4,4**		...
5							...
...	...	...	...	...	...	...	...

Como vemos, es como si estuviésemos dividiendo la imagen en cuadrículas de 2x2 píxeles y refrescando en cada una de ellas únicamente el píxel que ocupa la primera posición dentro de esa cuadrícula.

Si ahora nos fijamos en los valores de las direcciones de estos píxeles, nos fijamos que siguen la siguiente secuencia:

Imagen original: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ...

Imagen mosaico: 0, 0, 2, 2, 4, 4, 6, 6, 8, 8, ...

Es decir, viéndolo en binario, lo que estamos haciendo es “anular” el último bit de la dirección colocándolo siempre a cero.

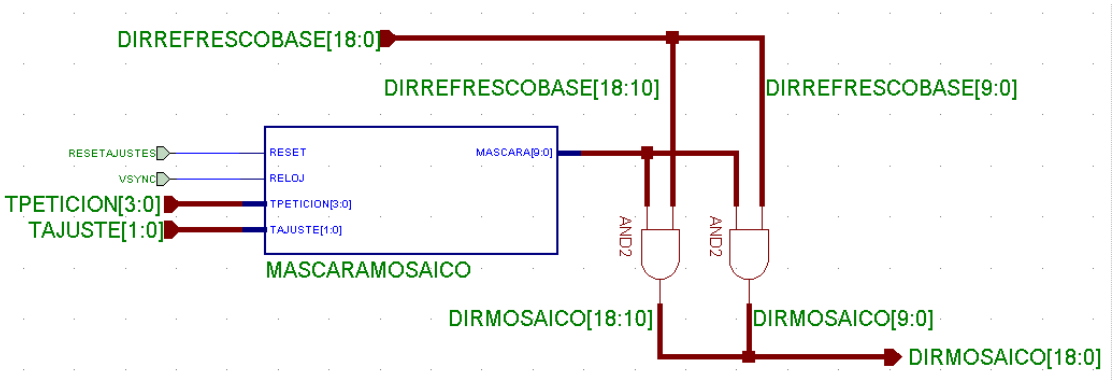
Una manera muy sencilla de conseguir esto es usar un registro que hace las funciones de “máscara” y que tiene todos sus bits a uno, salvo el último a cero y que, al hacer una “and” lógica con la dirección de refresco, nos anula ese último bit, dándonos la dirección que usaremos realmente para refrescar nuestra imagen. Obviamente, debido al mapa de memoria que estamos usando, tenemos que aplicarlo por separado a las dos componentes en que estamos dividiendo siempre la dirección de la memoria.

Haciéndolo así, éste será un efecto que se puede realizar en tiempo real sin ningún problema, pues se realiza únicamente una conversión de las direcciones que

utilizamos para refrescar desde la memoria (siempre del banco izquierdo) y con unos requisitos de hardware bastante pequeños.

Además, haciéndolo de esta manera, para modificar el factor del efecto, únicamente tendremos que actuar sobre el número de ceros que tiene la máscara. Para aumentar el factor del mosaico, reduciendo la resolución de la imagen, tendremos que desplazar la máscara hacia la izquierda, introduciendo ceros y, para disminuir el factor, aumentando la resolución, tendremos que realizar desplazamientos a la derecha, introduciendo unos.

El esquema de cómo lo hemos implementado sería el siguiente:



Como vemos, tenemos la máscara en un registro de desplazamiento que modifica su valor en función de si queremos aumentar o disminuir el factor y, continuamente estamos haciendo una “and” lógica con las dos componentes de la dirección de refresco base para anular (poner a cero) el número de bits que corresponda y que vendrá determinado por el número de ceros que tenga la máscara.

3.5.9.4. Zoom

Con el zoom pretendemos **modificar el tamaño de la imagen que estamos visualizando**.

Para ello, lo único que tenemos que hacer es refrescar varias veces, en función del factor de zoom, cada uno de los píxeles de la imagen original. Obviamente, como el tamaño de pantalla sigue siendo el mismo, al aumentar el tamaño de la imagen, hay una porción de ella que se sale fuera y no se visualiza.

Un ejemplo gráfico de cómo se aplicaría el zoom con un factor "x2" sería el siguiente:

Imagen Original

	0	1	**2**	3	**4**	5	...
0	**0,0**	0,1	**0,2**	0,3	**0,4**	0,5	...
1	1,0	1,1	1,2	1,3	1,4	1,5	...
2	**2,0**	2,1	**2,2**	2,3	**2,4**	2,5	...
3	3,0	3,1	3,2	3,3	3,4	3,5	...
4	**4,0**	4,1	**4,2**	4,3	**4,4**	4,5	...
5	5,0	5,1	5,2	5,3	5,4	5,5	...
...	...	...	...	...	...	...	...

Imagen Zoom x2

	0	1	**2**	3	**4**	5	...
0	**0,0**		**0,1**		**0,2**		...
1							...
2	**1,0**		**1,1**		**1,2**		...
3							...
4	**2,0**		**2,1**		**2,2**		...
5							...
...	...	...	...	...	...	...	...

Como vemos, es como si el valor de las direcciones estuviese avanzando a la mitad de velocidad, con lo que obtenemos dos veces cada uno de los píxeles.

Si ahora nos fijamos en los valores de las direcciones de estos píxeles, nos fijamos que siguen la siguiente secuencia:

Imagen original: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ...
Imagen mosaico: 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, ...

Es decir, viéndolo en binario, lo que estamos haciendo es un desplazamiento de la dirección un bit a la derecha, despreciando el bit menos significativo.

Cuantos mayor sea el número de desplazamientos, mayor será el factor aplicado y, si el desplazamiento lo hacemos en sentido contrario, hacia la izquierda, lo que hacemos el reducir el factor aplicando un "x1/2".

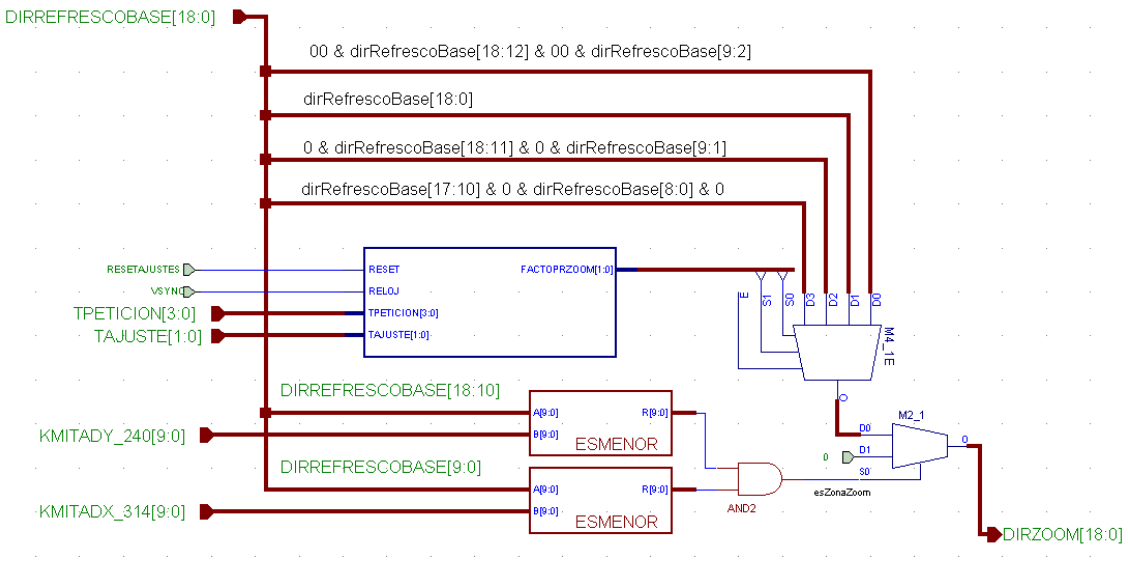
En principio, podría parecer que una manera sencilla de hacer esto es utilizar un registro de desplazamiento que, en función del número de bits que se desplacen, nos da uno u otro factor de zoom. Sin embargo, tenemos un problema: este desplazamiento supondría, al menos, un ciclo, con el retraso que esto acarrea al resto del sistema.

Por lo tanto, teniendo en cuenta que al implementarlo con desplazamiento los factores que vamos a obtener son potencias de dos, no tiene mucho sentido contemplar más allá del factor 4, con lo que hemos optado por realizar continuamente los cálculos de la dirección que se correspondería a cada uno de los cuatro factores que hemos contemplado: x1/2, x1, x2, x4 y seleccionar el que corresponda mediante un multiplexor que estará gobernado por el valor que tengamos en un registro.

Además, teniendo en cuenta que un factor de reducción de “x1/2” hace que queden tres cuartas partes de la imagen libres, tenemos que realizar en paralelo la comprobación de si estamos dentro del cuadrante superior izquierdo, en el que representaremos la imagen reducida o no. Esta condición únicamente se tendrá en cuenta en el caso de un zoom de reducción y lo que hace es anular la dirección, consiguiendo que esa parte aparezca en negro.

Haciéndolo así, éste será un efecto que se puede realizar en tiempo real sin ningún problema, pues se realiza únicamente una conversión de las direcciones que utilizamos para refrescar desde la memoria (siempre el banco izquierdo) y con unos requisitos de hardware bastante pequeños.

El esquema de cómo lo hemos implementado sería el siguiente:



Como vemos, el factor del zoom lo tenemos almacenado en un registro que nos permite seleccionar mediante un multiplexor el valor de la dirección que corresponda. En paralelo estamos realizando los cálculos de si estamos en zona de zoom o no y que nos permiten anular la dirección previamente seleccionada colocándola a cero.

3.5.9.5. Reflejo

Con el reflejo, lo que pretendemos es **obtener una imagen especular, es decir, la que obtendríamos al girar la imagen original sobre los ejes X (eje horizontal) y/o Y (eje vertical)**. Obviamente, al estar tratando imágenes en dos dimensiones, las posibilidades son cuatro: normal, 180 grados sobre el eje X, 180 grados sobre el eje Y, 180º sobre ambos ejes.

Si realizamos los cálculos pertinentes para cada uno de los casos, nos damos cuenta de que lo único que estamos haciendo es modificar la componente de la dirección que se corresponde con el eje que giramos y que esta modificación únicamente consiste en restarle al tamaño máximo de la pantalla para esa dimensión, el valor del píxel que estamos refrescando.

Es decir, si nuestra dirección de refresco base es:

Y	&	X
dirRefrescoBase[18..10]	&	dirRefrescoBase[9..0]

Los valores que obtendríamos para cada una de las cuatro posibilidades serían:

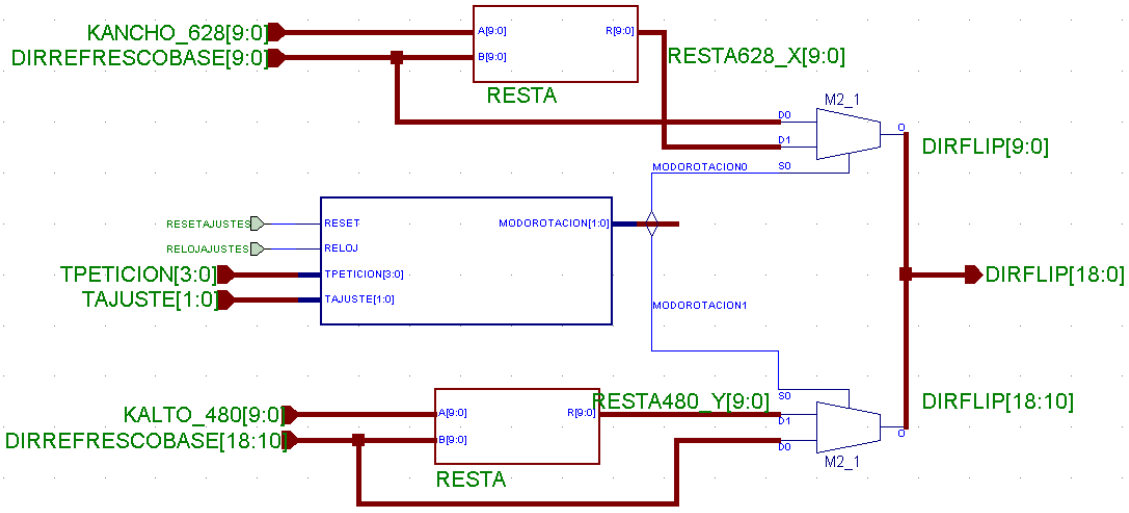
	DirRefrescoBase		Modo Flip	
	Componente Y	**Componente X**	**Modo(1)**	**Modo(0)**
Normal	Y	X	0	0
Horizontal	Y	Kancho_628 - X	0	1
Vertical	Kalto_480 - Y	X	1	0
Horiz.-Vert.	Kalto_480 - Y	Kancho_628 - X	1	1

En la parte de la derecha de la tabla, tenemos la codificación que hemos usado para cada uno de los modos de reflejo. La elección de estos valores no es casual sino que obedece a la necesidad de que éstos nos puedan servir para elegir en cada caso si la componente de la dirección del efecto es la de base o la resta.

Así, lo que hacemos es realizar continuamente y en paralelo las restas y en función del valor de los dos bits usados para codificar el modo del efecto, seleccionamos uno u otro. Para la componente X, el bit de selección es el menos significativo y pala la Y, el más significativo.

Por lo tanto, éste es un efecto que también podemos realizar en tiempo real, pues únicamente estamos haciendo operaciones sobre la dirección que utilizamos para refrescar la imagen de memoria (siempre desde el banco izquierdo).

El esquema de cómo lo hemos implementado sería el siguiente:



Como vemos tenemos almacenado el modo en el que nos encontramos en un registro de dos bits que nos permiten seleccionar entre las resta o la dirección base a la hora de componer la dirección final del efecto.

3.5.9.6. Rotación

Con este efecto, **obtenemos la imagen resultante de realizar una rotación de la imagen original sobre el eje Z, es decir, el eje que estaría perpendicular a la pantalla del monitor**.

Hemos contemplado los giros a intervalos de 90º por lo que las posibilidades son cuatro: 0º, 90º, 180º y 270º. Los ángulos de giro están medidos en sentido horario.

Si realizamos los cálculos pertinentes en función de la dirección de refresco base para cada una de las componentes por separado, obtendríamos una tabla como la siguiente:

Y	&	X
dirRefrescoBase[18..10]	&	dirRefrescoBase[9..0]

	DirRefrescoBase		Modo Rotación	
	Componente Y	**Componente X**	**Modo(1)**	**Modo(0)**
0º	Y	X	0	0
90º	Kalto_480 - X	Y	0	1
180º	Kalto_480 - Y	Kancho_628 - X	1	0
270º	X	Kancho_628 - Y	1	1

En este caso, vemos que para cada una de las componentes tenemos cuatro valores diferentes, por lo que no nos ha sido posible simplificar el esquema de selección y tenemos que utilizar para cada una de ellas un multiplexor 4 a 1 gobernado por los dos bits que usamos para codificar el modo de rotación.

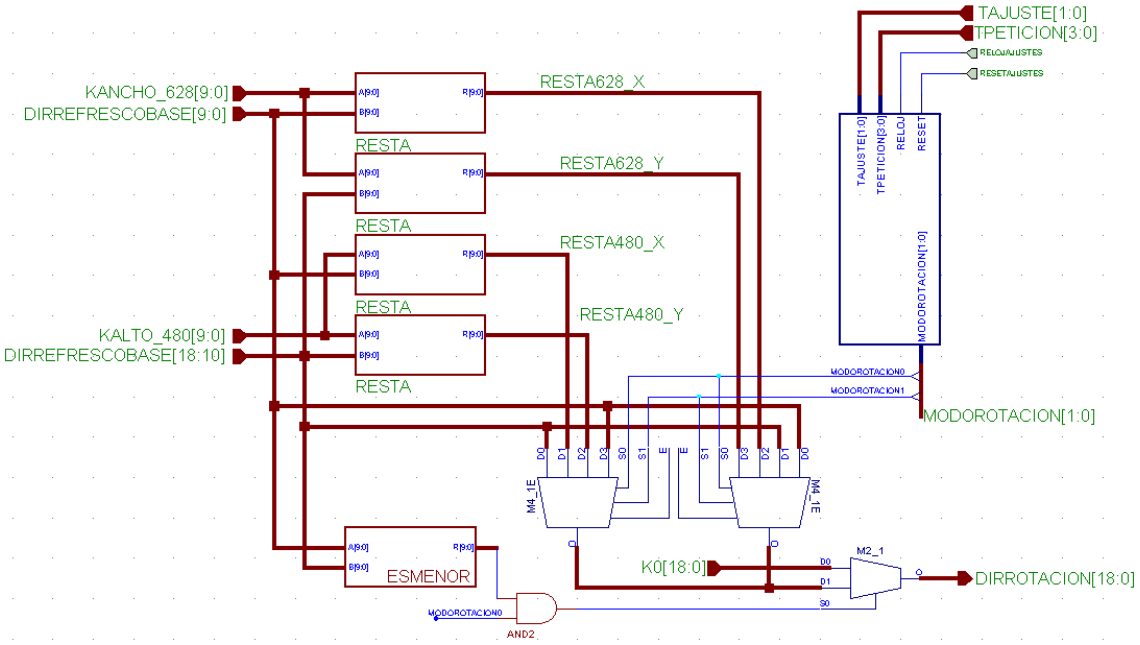
Vemos que el esquema de obtención de la dirección es muy similar al que hemos usado en el reflejo y, de hecho, las restas “KAlto_480 – Y” y “KAncho_628 – X”, son las mismas. Es decir, partiendo de los elementos que necesitamos para el reflejo, únicamente tenemos que añadir dos nuevas restas y los dos multiplexores para elegir la componente que corresponda.

Sin embargo, en el caso de la rotación, tenemos que contemplar una cosa más y es que en las de 90º y 270º, como hemos intercambiado los ejes de la imagen, las dimensiones de ésta no coinciden con las de la pantalla, es decir, estamos representando el alto de la imagen a lo largo del eje X de la pantalla, y el ancho de la imagen a lo largo del eje Y de la pantalla. Por lo tanto, en ambos casos habrá una parte de la pantalla (exactamente “KAncho_628 – KAlto_480” píxeles) que quedará en negro.

La manera más sencilla de hacer esto es mediante una comparación entre el valor que estamos usando para refrescar la imagen y el valor del alto de la pantalla. En caso de que sea mayor, tendremos que ponerlo negro. Para ello, hemos optado por anular la dirección que estamos usando para refrescar y ponerla a cero.

Nuevamente, estamos ante un efecto que se puede realizar en tiempo real sin ninguna complicación pues únicamente estamos haciendo cálculos sobre la dirección que usamos para refrescar la imagen de la memoria (siempre del banco izquierdo).

El esquema de cómo lo hemos implementado sería el siguiente:



Como vemos, realizamos en paralelo las cuatro restas y en función de los dos bits con los que codificamos el tipo de rotación, seleccionamos los valores que compondrán la dirección que usaremos. Por otro lado, y también en paralelo, se está haciendo la comparación para saber si estamos fuera de zona y en una rotación de 90º ó 270º (en cuyo caso vale uno el bit menos significativo) para poder anular la dirección que finalmente usaremos.

3.5.9.7. Pip

Con el Pip, lo que conseguimos es **visualizar simultáneamente la imagen que se está generando y una imagen estática que previamente hemos capturado y que está almacenada en memoria**. Esta última se muestra en uno de los cuatro cuadrantes de la pantalla y está superpuesta sobre la imagen que se está generando en ese momento.

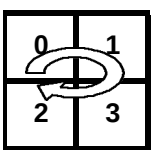
La dificultad de este efecto radica en tres puntos clave:

1. Accedemos a los dos bancos de la memoria: al izquierdo para obtener los píxeles correspondientes a la imagen que se está generando en ese momento, y al derecho, para obtener la imagen almacenada que mostraremos en el cuadrante que corresponda.
2. La posición del cuadrante sobre el que mostramos la imagen estática no es fijo, sino que es seleccionable por el usuario.
3. Hay que realizar un reescalado de la imagen a mostrar en el banco derecho porque tenemos que mostrar una pantalla completa en un cuarto de ella.

El primero de ellos se soluciona fácilmente comparando el valor de la dirección que estamos usando para refrescar con los valores que tenemos almacenados en dos registros y que nos dan el origen del cuadrante del pip. Para obtener las dimensiones del cuadrante, únicamente tenemos que tener en cuenta que las dimensiones son justo la mitad de la pantalla y que tenemos representadas con constantes, por lo que para saber si estamos dentro o no, únicamente tenemos que hacer una doble comparación para cada una de las dimensiones. El valor de esta señal, que únicamente valdrá uno dentro del cuadrante, se le pasa a la máquina de estados para que pueda discriminar si pasa a un estado de refresco del banco izquierdo (cuando esté fuera del cuadrante y la señal valga cero) o de refresco del banco derecho (cuando esté dentro del cuadrante y la señal valga uno).

El segundo problema viene porque, al poder cambiar el cuadrante, los valores de esos dos registros han de cambiar en función de lo que quiera el usuario. Sin embargo, puesto que únicamente tenemos cuatro posiciones, los problemas se reducen bastante. Si además, tenemos un poco de cuidado a la hora de implementarlo, vemos que se puede simplificar enormemente.

Para identificar el cuadrante, numeraremos cada uno de ellos en el sentido de las agujas del reloj pero, en vez de utilizar la numeración binaria normal, usaremos un código Gray, que nos facilitará mucho las cosas a la hora de seleccionar los límites. Haciéndolo de esta manera, nos damos cuenta de que cada una de las componentes del origen del cuadrante depende únicamente de uno de los bits del contador por lo que sólo necesitaremos multiplexores de dos a uno, gobernados cada uno por uno de los bits del contador. En la siguiente tabla se ve claramente.

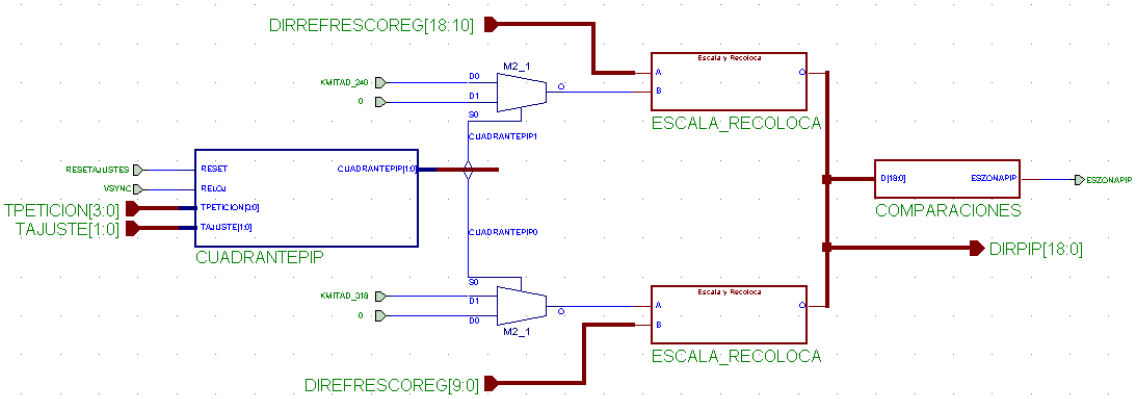


Cuadrante Pip			Origen Y	Origen X
Arriba-Izqda.	**0**	**00**	0	0
Arriba-Drcha.	**1**	**01**	0	KMitadX_314
Abajo-Izqda.	**2**	**10**	KMitadY_240	0
Abajo-Drcha.	**3**	**11**	KMitadY_240	KMitadX_314

Es decir, el valor de la componente Y del origen depende del bit más significativo del contador (KMitadY_240 para los de abajo, 2 y 3), y el de la componente X, del bit menos significativo del contador (KMitadX_314 para los de la derecha, 1 y 3).

El tercero, tampoco trae demasiados problemas, pues únicamente tenemos que tener en cuenta que las direcciones que usemos para acceder a la imagen estática han de producir un efecto de reducción "x1/2" sobre ella y que, como ya hemos visto en el zoom, esto se reduce a realizar un desplazamiento de un bit a la izquierda.

El esquema de cómo lo hemos implementado es el siguiente:



Como vemos, el cuadrante lo controlamos mediante un contador Gray módulo cuatro que se incrementa con las pulsaciones del usuario. En función de los bits que lo codifican seleccionamos las componentes del origen del cuadrante sobre el que nos toca escribir. Finalmente, con estos dos valores y con las componentes de la dirección de refresco base, hacemos un reescalado y una recolocación para obtener la dirección que usaremos para acceder al banco derecho de la memoria. En función del valor que tome la señal que nos indica si estamos dentro del cuadrante o no, la máquina de estados usará esta dirección o no.

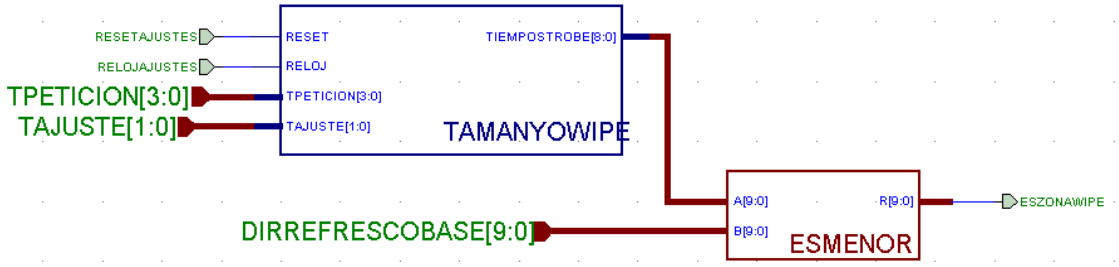
3.5.9.8. Wipe

Con el wipe, **mostramos simultáneamente en la pantalla dos imágenes: la que se está generando en ese momento en la mitad izquierda, y la que hay almacenada en memoria en la mitad derecha**. La línea que divide ambas imágenes se puede desplazar a la izquierda y la derecha para aumentar y disminuir el espacio dedicado a cada una de ellas.

En principio, el único requerimiento de este efecto es el tener un registro en el que se almacena la posición que delimita ambas partes de la pantalla para que, al hacer una comparación entre el contenido de éste y el valor de la componente X de la dirección que estamos usando para el refresco, podamos saber si estamos en una u otra zona. Con el resultado de esta comparación, se genera una señal que vale uno en caso de que estemos en la zona dedicada a la imagen estática y que se le pasa a la máquina de estados para que pueda discriminar si tras un estado de escritura para a un estado de refresco del banco izquierdo de la memoria para mostrar la imagen que se está generando en ese momento (en caso de que la señal valga cero), o pase a un estado de refresco del banco de la derecha, en el caso de que la señal valga uno.

Por supuesto, éste es también un efecto que podemos ir realizando en estricto tiempo real, pues los requisitos harware son muy sencillos.

El esquema que hemos usado en la implementación es el siguiente:



Cómo vemos, únicamente tenemos el registro en el que almacenamos el valor que nos indica cuándo estamos en una u otra zona y que varía con las pulsaciones adecuadas del teclado y una sencilla comparación entre este valor y la componente X de la dirección de refresco que estamos usando para acceder a la memoria.

3.5.9.9. Estrobe

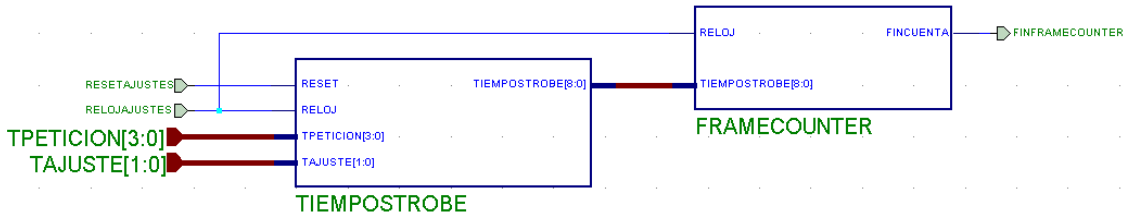
Con el estrobe, lo que conseguimos es **variar la frecuencia con que se actualiza la imagen que se visualiza**. Es decir, la imagen se congela y después se mantiene estable durante un intervalo de tiempo antes de volver a actualizarse.

La manera más sencilla de implementar este efecto es realizar cada cierto tiempo una captura de la imagen en curso y luego, durante todo el intervalo, refrescar la imagen que acabamos de captura en lugar de la que se esté generando.

Para controlar el intervalo de tiempo, usaremos un contador que se irá decrementando con cada imagen (frame) refrescado y en el momento que llegue a cero, dará una señal para que se realice una captura de la imagen en curso y se recargará con el valor que tengamos almacenado en un registro. La manera de variar este intervalo se consigue modificando el valor del registro del que cogemos el valor cuando iniciamos un nuevo intervalo.

En principio, esto sería todo lo necesario para poder implementarlo pues ahora, lo único que queda es que la máquina de estados que alterna los estados de escritura y refresco se encargue de hacer las transiciones al estado de refresco del banco derecho durante todo el intervalo de tiempo, o al de escritura en el banco derecho en el caso de que se haya acabado el temporizador y tengamos que realizar una nueva captura.

El esquema de la implementación sería el siguiente:



Como vemos, el “frameCounter” es el temporizador que nos da la señal que irá a la máquina de estados y que obtiene su valor de un registro que modificaremos aumentando o disminuyendo su valor en función de que queramos disminuir la frecuencia de captura o aumentarla respectivamente.

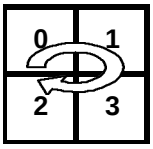
3.5.9.10. Estrobe Cíclico

Con este efecto, lo que pretendemos es **realizar un estrobe solo que, en lugar de visualizar únicamente la última imagen capturada, tenemos en pantalla las cuatro últimas. Éstas se irán sobrescribiendo cíclicamente en el sentido de las agujas del reloj y se mostrarán únicamente en unos de los cuatro cuadrantes de la pantalla**.

En principio, éste es un efecto complejo, pero si aprovechamos los razonamientos que hemos tenido que hacer para otros, nos damos cuenta que el comportamiento es muchas veces similar, por lo que trataremos de aprovecharlos.

En primer lugar, al igual que el estrobe, estamos refrescando durante cierto tiempo una imagen que actualizamos cada vez que el temporizador se termina. Esto nos lleva a aprovechar las estructuras que hemos usado para el estrobe para la implementación del temporizador, con lo que ya tenemos esto solucionado.

En segundo lugar, al contrario que en el estrobe, no escribimos sobre toda la pantalla, sino que únicamente sobre el cuadrante que corresponda y que irá cambiando con cada captura. Esto nos lleva a implementar una estructura de control del cuadrante similar a la que usamos en el pip, solo que ahora, el cambio de un cuadrante a otro no depende del usuario, sino del propio temporizador. Una vez conocido el cuadrante sobre el que nos toca escribir, ya sabemos donde se sitúa nuestro origen de las direcciones de escritura. Si tenemos un poco de cuidado en la codificación de los cuadrantes, nos damos cuenta que al numerarlos en sentido de las agujas del reloj y usando una codificación Gray, los valores de cada una de las coordenadas dependen únicamente de uno de los bits del contador (por lo que únicamente necesitaremos multiplexores de dos a uno) tal y como queda reflejado en la siguiente tabla:



Cuadrante StrobeC			Origen Y	Origen X
Arriba-Izqda.	**0**	**00**	0	0
Arriba-Drcha.	**1**	**01**	0	KMitadX_314
Abajo-Izqda.	**2**	**10**	KMitadY_240	0
Abajo-Drcha.	**3**	**11**	KMitadY_240	KMitadX_314

Es decir, el valor de la componente Y del origen depende del bit más significativo del contador (KMitadY_240 para los de abajo, 2 y 3), y el de la componente X, del bit menos significativo del contador (KMitadX_314 para los de la derecha, 1 y 3).

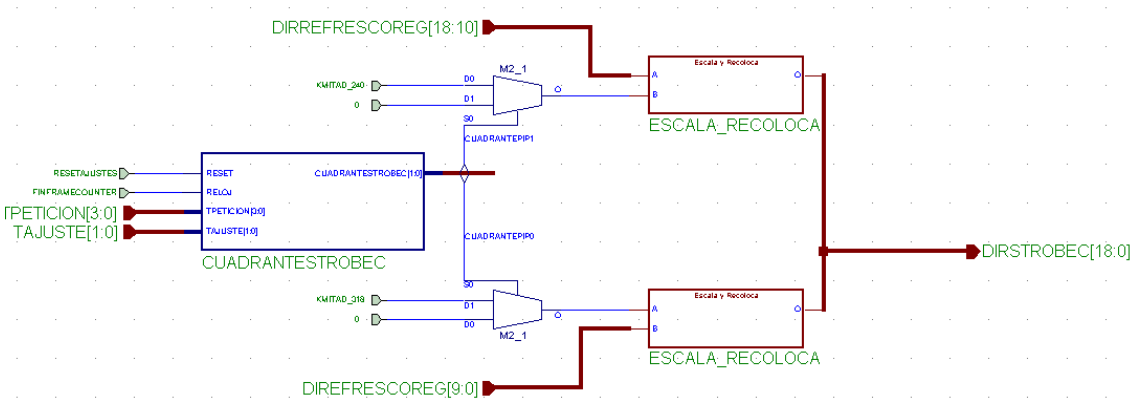
En tercer lugar, tenemos que tener en cuenta que, como estamos escribiendo sobre un cuadrante, no podemos usar directamente la dirección que nos llega, sino que tenemos que modificarla haciendo una reducción de la imagen para que quepa completa en ese cuadrante, con lo que estamos haciendo un proceso similar al del zoom con factor x1/2.

Obviamente, aunque el razonamiento sea similar al utilizado en estos tres casos, los componentes no son reutilizables porque son efectos totalmente independientes y diferentes.

Por lo tanto, el proceso que sigue el estrobe cíclico cada vez que toca una escritura es el siguiente:

1. Obtención del origen de coordenadas del cuadrante sobre el que vamos a escribir en función del valor de un contador que va variando cada vez que el temporizador finaliza
2. Reescalado de la imagen aplicando una reducción para adaptarla al tamaño de un cuadrante
3. Recolocación de la dirección en función del origen de coordenadas del cuadrante sobre el que toca escribir
4. Escritura de la imagen

El esquema de la implementación que hemos hecho es el siguiente:



Como vemos, el cuadrante lo controlamos mediante un contador Gray módulo cuatro que se incrementa cíclicamente cada vez que termina el temporizador. En función de los bits que lo codifican seleccionamos las componentes del origen del cuadrante sobre el que nos toca escribir. Finalmente, con estos dos valores y con las componentes de la dirección de refresco base, hacemos un reescalado y una recolocación para obtener la dirección sobre la que escribiremos.

En este caso, al contrario que en otros efectos en los que trabajamos con direcciones modificadas, no es necesario que comprobemos si estamos en zona o no porque sólo generamos direcciones válidas.

3.5.10. Interfaz Teclado - Máquina de Estados

Este módulo tiene como entradas el reloj general , el reloj del teclado y los datos que se reciben del teclado. Como salida indica el estado de la máquina de efectos en función de la última tecla pulsada y el tipo de ajuste (positivo, negativo o sin ajuste).

Este módulo está formado por los siguientes componentes:

- Comprobador de tecla: comprueba las pulsaciones
- Registro de ajustes auxiliar: se carga cada vez que haya una pulsación de una tecla de ajustes, es decir, cada vez que se pulsa la tecla '+' o la tecla '-'. Este registro se resetea tras la señal de sincronismo vertical (vsync), para evitar que se lea más de una vez su valor.
- Registro de ajustes: guarda el ajuste en el que se queda el registro. Funciona con la señal de sincronismo vertical (vsync).
- Registro de estado auxiliar: carga el valor del estado que se corresponde con la tecla pulsada.
- Registro de estado: guarda el estado en el que se queda tras haber un flanco de subida en la señal de sincronismo vertical (vsync).

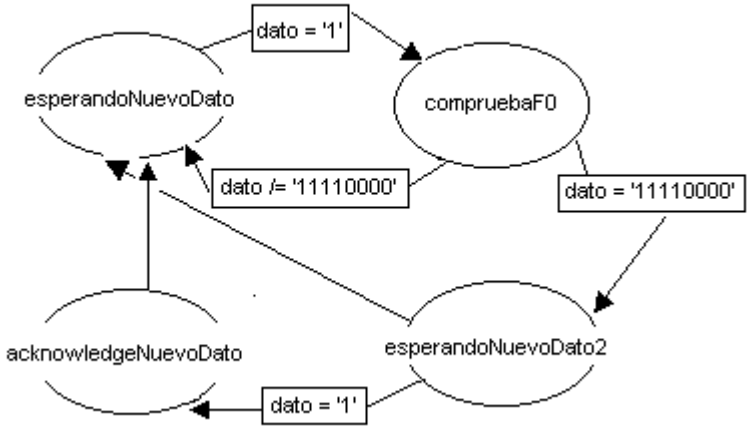
Para más detalle consultar "*interfazTecladoMaquinaEstados.vhd*" en el apéndice.

3.5.11. Comprobador Tecla

Este módulo actúa por encima del módulo que hace de interfaz con el teclado PS/2. Dadas las señales de reset, reloj, reloj del teclado y un dato del teclado devuelve pulsación (indica que ha habido una pulsación de una tecla cuando ya se ha dejado de pulsar dicha tecla) y tecla (devuelve 8 bits que indican el código de la tecla pulsada).

El módulo tiene un componente que es *interfaz teclado PS2*, y funciona mediante una máquina de 4 estados que lleva el control. Los estados son los siguientes:

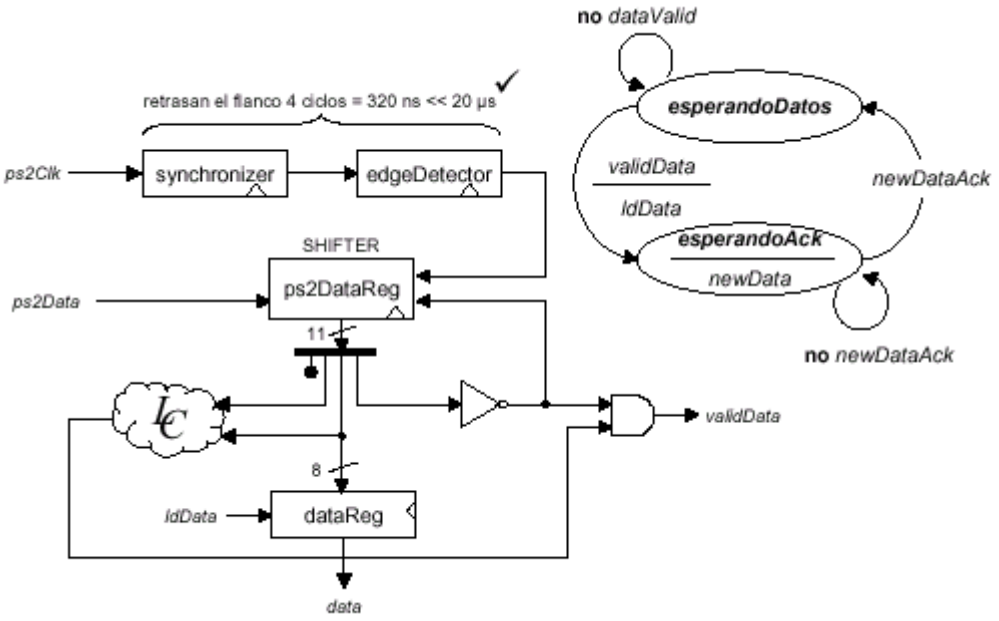
- esperandoNuevoDato: está en este estado antes de recibir un dato
- compruebaF0: comprueba si el código recibido es F0.
- esperandoNuevoDato2: si el código recibido es F0, entonces espera el código de depresión
- acknowledgeNuevoDato: estado de confirmación de la tecla recibida.



3.5.12. Interfaz teclado PS/2

Implementa un interfaz con un teclado PS/2. Recibe del teclado su reloj y una señal de datos. Como la comunicación con el teclado es mediante una única línea, tiene que ir componiendo el código de la tecla pulsada para poder suministrarlo con 8 bits. Además dispone de una señal para indicar que se ha producido una nueva pulsación, y una señal para que se le indique que ya se muestreó el valor de la pulsación anterior.

La implementación de este módulo es la siguiente:



Para más detalle consultar en el apéndice "*interfazTecladoPS2.vhd*".

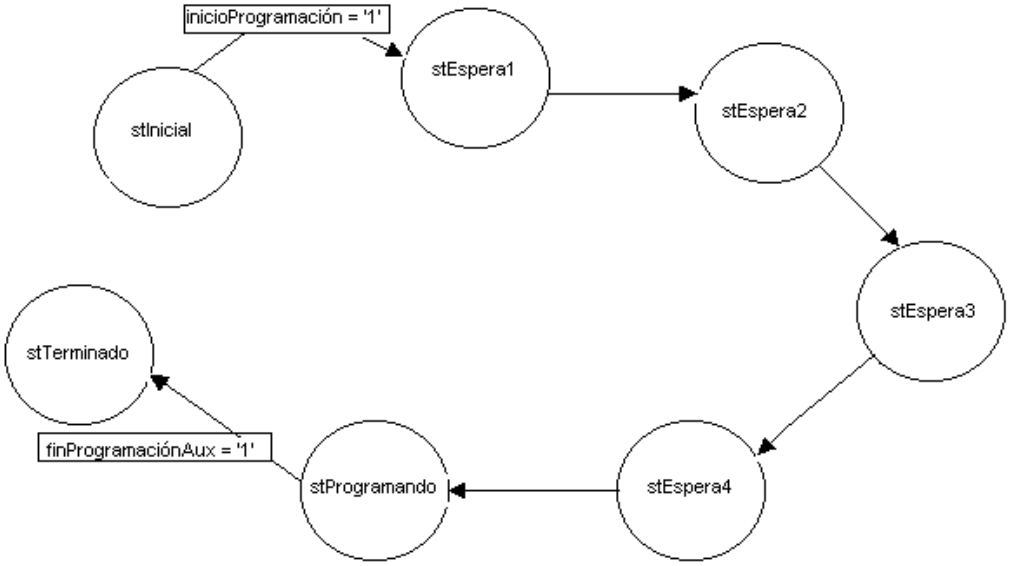
3.5.13. Programa RamDac

Este módulo actúa por encima del auténtico programador y nos permite programar el RamDac para que funcione en color real en el formato 5:5:5 funcionando con un solo flanco o en escala de grises.

Este módulo tiene el componente *programa RamDac true color*. Está controlado por una máquina de estados con los siguientes estados:

- stInicial : permanece en este estado hasta que se ponga en alta la señal de inicioProgramación.
- stEspera1: ciclo de espera.
- stEspera2: ciclo de espera.
- stEspera3: ciclo de espera.
- stEspera4: ciclo de espera.
- stProgramando: permanece en este estado hasta que se termine de programar
- stTerminado: es el estado final. Se pasa a este estado cuando hemos terminado de programar el RamDac.

En la siguiente figura se puede ver el diagrama de estados:

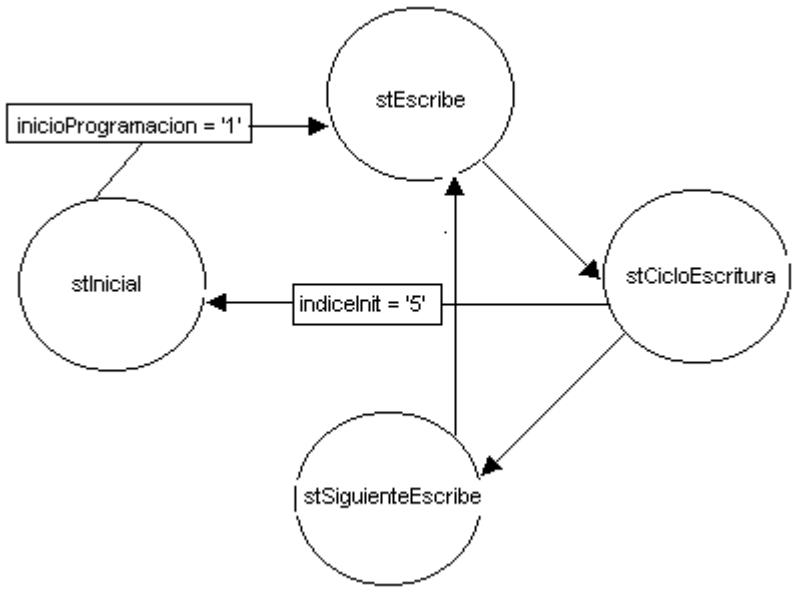


3.5.14. Programa RamDac True Color

El módulo programa el RamDac para que funcione con color real en el formato 5:5:5 con un solo flanco(single edge). Lo que hay que hacer es configurar adecuadamente el registro de comandos A. Para comenzar a programar se pone a uno la señal inicioProgramación y esperamos hasta que se ponga en alta la señal finProgramación.

Este módulo tiene un controlador con los siguientes estados:

- stInicial: permanece en este estado hasta que la señal inicioProgramación se pone en alta.
- stEscribe: desde este estado se pasa a stCicloEscritura.
- stCicloEscritura: si se ha terminado de programar se vuelve al estado stInicial, y si no se pasa al estado stSiguienteEscribe.
- stSiguienteEscribe: desde este estado siempre se pasa a stEscribe.



Consúltese "*programaRamDacTrueColor.vhd*" para información más detallada.

4. Conclusiones

La principal conclusión del proyecto es la que se sigue de la obtención de un prototipo utilizable. Se ha comprobado que la plataforma de desarrollo elegida es suficientemente potente para obtener una implementación que cumpla con los propósitos especificados.

El afán que hemos perseguido ha sido, antes que nada, la exploración de las posibilidades del entorno de desarrollo y el conseguir una visión práctica y real de lo que puede suponer una aplicación tecnológica concreta.

Esta proyección investigadora implica que no se haya pretendido obtener productos terminados de menores pretensiones, que pueden obtenerse en un tiempo menor (por ejemplo, empleando menor profundidad de color).

Las pregonadas ventajas del hardware reconfigurable, en lo referente a amplia versatilidad y moderado tiempo de desarrollo, se han probado ciertas en buena medida. Estas características han hecho posible que pudieran considerarse alternativas muy distintas para cada uno de los frentes de diseño.

Particularmente, la aceleración del ciclo de desarrollo ha permitido la realización de miniprototipos que se han usado continuadamente para medir la capacidad del sistema y orientar consecuentemente las directrices del diseño. Sirvan como ejemplo los desarrollos intermedios que nos demostraron la correcta temporización de la memoria o el funcionamiento básico del RamDac.

Corresponde, igualmente, señalar las limitaciones. La propia generalidad de la placa impone restricciones cuando su uso se reduce a las prestaciones de video, como en la compartición de líneas del RamDac. Indudablemente, la facilidad para prototipar es contraria a la obtención de implementaciones que persigan la eficiencia en usos particulares. Cabe reseñar que también existen placas de prototipado para fines específicos como el procesamiento de video.

La principal inconveniencia de cara a conseguir un producto terminado está en ciertas interferencias entre módulos conceptualmente separados. Nuestra experiencia ha constatado que cambios locales introducidos de una versión a otra pueden producir distorsiones en otros cálculos independientes o paralelos. Ello se justifica por la diferente disposición de las conexiones internas de la FPGA que derivan automáticamente las herramientas de apoyo, que afecta, por ejemplo al retardo de los caminos combinacionales.

Para el desarrollo de proyectos de una cierta magnitud consideramos fundamental un conocimiento exhaustivo de los pormenores del vhdl sintetizable. Toda riqueza expresiva tiene que quedar condicionada a las estructuras que deriven una implementación como la que el desarrollador tiene en mente. En este sentido, es deseable un conocimiento más profundo del paquete de herramientas de apoyo.

Así mismo, y aunque han quedado fuera de nuestro alcance, es muy conveniente el estudio de aquellas directivas que pueden especificar heurísticas de colocación de componentes en la placa, de forma que se pueda estrechar la variabilidad entre la descripción y la realidad física correspondiente.

Respecto a las fases de ejecución del proyecto, el mayor esfuerzo corresponde a coordinar la acción de todos los componentes. El sincronismo tiene especial relevancia puesto que el RamDac, las pulsaciones de teclado (netamente asíncronas), la generación de la simulación de video, y las escrituras en la memoria llevan sus propios ritmos de funcionamiento, pero deben, como es obvio, actuar aunadamente.

La ideación de la arquitectura básica de comunicación entre estos componentes ha sido el hito principal del desarrollo. Se ha conseguido una arquitectura básica robusta, con fluidez en el traspaso de la imagen y, como característica fundamental, muy extensible mediante cajas negras que pueden implementar, con sencillez, una rica variedad de efectos. Ésta ha sido la fase más costosa del desarrollo, con aproximadamente un 70% de esfuerzo. A la documentación previa e implementaciones finales les corresponde un 15% a cada una.

Finalmente, esta misma cualidad introduce una de las ampliaciones que con más claridad podrían acometerse en futuros desarrollos. La inclusión de nuevos efectos (por ejemplo, anular determinadas componentes de color) o la variación de algunos de los existentes (un zoom con foco variable) resultan notablemente accesibles. La búsqueda de mayor eficiencia en las implementaciones y el empleo de una señal de video real mediante la conexión de un codec, son otras de las extensiones más naturales.

Apéndice: Código Fuente

video.vhd

```
--------------------------------------------------------------------------------
-- MODULO SUPERIOR DEL PROYECTO TELEVISION DIGITAL
--------------------------------------------------------------------------------
-- ENTIDAD: video
-- CONEXIONES:
-- PUERTOS ENTRADA:
--    relojGeneral: Reloj de entrada (50 MHz)
--    resetGeneral: Reset general (activo a baja)
-- OTROS PUERTOS (agrupados por funciones):
--    -- Senyales para la programacion del RamDac
--      WrPaleta (out): Senyal de escritura en la paleta (activa a baja)
--      RdPaleta (out): Senyal de lectura de la paleta (activa a baja)
--      RS[2..0] (inout): Lineas de seleccion de registro (register select)
--      datosPaleta[7..0] (inout): Linea bidireccional de datosPaleta con el RamDac
--    -- Senales de sincronizacion del RamDac
--      hSync (out): Sincronizacion horizontal (activa a baja)
--      vSync (out): Sincronizacion Vertical (activa a baja)
--      blanking (out): Senyal de blanking (activa a baja)
--      pixelClock (out): Reloj del RamDac (50 MHz)
--    -- Banco Izquierdo SRam
--      WEI (out): Write Enable de la SRAM (activa a baja)
--      OEI (out): Output Enable de la SRAM (activa a baja)
--      CSI (out): Chip Enable de la SRAM (activa a baja)
--      datosMemorial[15..0] (buffer): Buffer de datos con la SRAM
--      direccionMemorial[18..0] (out): Direccion de la memoria
--    -- Banco Derecho SRam
--      WED (out): Write Enable de la SRAM (activa a baja)
--      OED (out): Output Enable de la SRAM (activa a baja)
--      CSD (out): Chip Enable de la SRAM (activa a baja)
--      datosMemoriaD[15..0] (buffer): Buffer de datos con la SRAM
--      direccionMemoriaD[18..0] (out): Direccion de la memoria
--    -- Teclado
--      relojTeclado (in): Reloj del teclado
--      datosTeclado (in): Datos del teclado
--    -- Ethernet
--      ethernet (out): Capacitacion de Ethernet (activo a baja) (a uno para desactivarla)
--------------------------------------------------------------------------------
-- Contiene dos componentes: el programador del RamDac y la Maquina General
-- Se encarga de que se programe primero el RamDac y despues se comience a
--  funcionar con normalidad.
-- Para ello, va pasando por cuatro estados y dando las senyales de reset, inicio...
--  que correspondan
```

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

--------------------------------------------------------------------------------
-- Declaracion de la entidad
--------------------------------------------------------------------------------
entity video is
  port (
    relojGeneral: in std_logic;   -- Reloj de entrada (50 MHz)
    resetGeneral: in std_logic;   -- Reset general (activo a baja)

    -- Senyales para la programacion del RamDac
    WrPaleta: out STD_LOGIC;           -- Senyal de escritura en la paleta (activa a baja)
    RdPaleta: out STD_LOGIC;           -- Senyal de lectura de la paleta (activa a baja)
    RS: inout STD_LOGIC_VECTOR (2 downto 0); -- Lineas de seleccion de registro (register
select)
    datosPaleta: inout STD_LOGIC_VECTOR (7 downto 0);   -- Linea bidireccional de
datosPaleta con el RamDac

    -- Senales de sincronizacion del RamDac
    hSync: out std_logic;              -- Sincronizacion horizontal (activa a baja)
    vSync: out std_logic;              -- Sincronizacion Vertical (activa a baja)
    blanking: out std_logic;           -- Senyal de blanking (activa a baja)
    pixelClock: out std_logic;         -- Reloj del RamDac (50 MHz)

    -- Banco Izquierdo SRam
    WEI: out std_logic;               -- Write Enable de la SRAM (activa a baja)
    OEI: out std_logic;               -- Output Enable de la SRAM (activa a baja)
    CSI: out std_logic;               -- Chip Enable de la SRAM (activa a baja)
    datosMemorial: inout std_logic_vector(15 downto 0);     -- Buffer de datos con la SRAM
    direccionMemorial: out std_logic_vector(18 downto 0);     -- Direccion de la memoria

    -- Banco Derecho SRam
    WED: out std_logic;               -- Write Enable de la SRAM (activa a baja)
    OED: out std_logic;               -- Output Enable de la SRAM (activa a baja)
    CSD: out std_logic;               -- Chip Enable de la SRAM (activa a baja)
    datosMemoriaD: inout std_logic_vector (15 downto 0);   -- Buffer de datos con la SRAM
derecho
    direccionMemoriaD: out std_logic_vector (18 downto 0);   -- Direccion de la memoria del
banco derecho
```

```
    -- Teclado
    relojTeclado: IN std_logic;            -- Reloj del teclado
    datosTeclado: IN std_logic;             -- Datos del teclado

    -- Para iluminar los leds
    estadofsms: out std_logic_vector(4 downto 0);

    -- Switch para pausar la escrituras
    pausa: in std_logic;

    -- Ethernet
    ethernet: out std_logic              -- Capacitacion de Ethernet (activo a baja)
  );
end;
--------------------------------------------------------------------------------



--------------------------------------------------------------------------------
-- Declaracion de la arquitectura
--------------------------------------------------------------------------------
architecture video_arquitectura of video is



--------------------------------------------------------------------------------
-- Componente para programar el RamDac
--------------------------------------------------------------------------------
component programaRamDac
   port (
      relojProgramador: in STD_LOGIC;   -- Reloj de entrada (a 50MHz)
      resetProgramador: in STD_LOGIC;   -- Reset del circuito (activo a baja)
      inicioProgramacion: in STD_LOGIC; -- Senyal de inicio (activa a alta)
      finProgramacion: out STD_LOGIC;   -- Senyal de finalizacion (activa a alta)
      WrPaleta: out STD_LOGIC;           -- Senyal de escritura en la paleta (activa a baja)
      RdPaleta: out STD_LOGIC;           -- Senyal de lectura de la paleta (activa a baja)
      RS: inout STD_LOGIC_VECTOR (2 downto 0); -- Lineas de seleccion de registro (register
select)
      datosPaleta: inout STD_LOGIC_VECTOR (7 downto 0)   -- Linea bidireccional de
datosPaleta con el RamDac
   );
end component;
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
-- Componente para la maquina general
--------------------------------------------------------------------------------
component maquina
   port (
     -- Vendran de un modulo superior (no son las del sistema)
     resetMaquina: in std_logic;             -- senyal de inicializacion (activa a cero)
     relojMaquina: in std_logic;             -- reloj de entrada

     -- TECLADO
     relojTeclado: IN std_logic;           --Reloj del teclado
     datosTeclado: IN std_logic;          --Datos del teclado

     --Senales al RamDac
     hSync: out std_logic;               -- Sincronizacion horizontal (activa a baja)
     vSync: out std_logic;               -- Sincronizacion Vertical (activa a baja)
     blanking: out std_logic;            -- Senyal de blanking (activa a baja)

     --Banco Izquierdo SRam
     WEI: out std_logic;                -- Write Enable de la SRAM (activa a baja)
     OEI: out std_logic;                -- Output Enable de la SRAM (activa a baja)
     CSI: out std_logic;                -- Chip Enable de la SRAM (activa a baja)
     datosMemoriaI: inout std_logic_vector(15 downto 0);     -- Buffer de datos con la SRAM
     direccionMemoriaI: out std_logic_vector(18 downto 0);     -- Direccion de la memoria

     --Banco Derecho SRam
     WED: out std_logic;                -- Write Enable de la SRAM (activa a baja)
     OED: out std_logic;                -- Output Enable de la SRAM (activa a baja)
     CSD: out std_logic;                -- Chip Enable de la SRAM (activa a baja)
     datosMemoriaD: inout std_logic_vector (15 downto 0);   -- Buffer de datos con la SRAM
derecho
     direccionMemoriaD: out std_logic_vector (18 downto 0);   -- Direccion de la memoria del
banco derecho

     -- Para iluminar los leds
     estadofsms: out std_logic_vector(4 downto 0);

     -- Para pausar las escrituras en memoria
     pausa: in std_logic
    );
end component;
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
-- Declaracion de senyales
  -- Tipo para los estados de la FSM
  type tEstado is (stInicial, stProgramando, stFuncionando, stFuncionando2);
  signal estadoActual: tEstado;

  -- Senyales internas auxiliares
  -- Para la programacion del RamDac:
    signal resetProgramadorAux: std_logic;
    signal inicioProgramacionAux: std_logic;
    signal finProgramacionAux: std_logic;
  -- Para la maquina general:
    signal resetMaquinaAux: std_logic;
  -- Para el pixelClock:
    signal pixelClockAux: std_logic;
--------------------------------------------------------------------------------

begin


--------------------------------------------------------------------------------
-- Instanciacion del programador del RamDac
--------------------------------------------------------------------------------
programadorDelRamDac: programaRamDac
   port map (
      relojProgramador => relojGeneral,             -- Reloj de entrada (a 50MHz)
      resetProgramador => resetProgramadorAux,     -- Reset generado internamente (activo a
baja)
      inicioProgramacion => inicioProgramacionAux, -- Senyal de inicio (activa a alta)
      finProgramacion => finProgramacionAux,       -- Senyal de finalizacion (activa a alta)
      WrPaleta => WrPaleta,        -- Senyal de escritura en la paleta (activa a baja)
      RdPaleta => RdPaleta,      -- Senyal de lectura de la paleta (activa a baja)
      RS => RS,                 -- Lineas de seleccion de registro (register select)
      datosPaleta => datosPaleta -- Linea bidireccional de datosPaleta con el RamDac
   );
--------------------------------------------------------------------------------



--------------------------------------------------------------------------------
-- Instanciacion de la maquina general
--------------------------------------------------------------------------------
maquinaGeneral: maquina
   port map (
     -- Vendran de un modulo superior (no son las del sistema)
     resetMaquina => resetMaquinaAux,      -- senyal de inicializacion (activa a cero)
     relojMaquina => relojGeneral,               -- reloj de entrada
     -- TECLADO
     relojTeclado => relojTeclado,           --Reloj del teclado
     datosTeclado => datosTeclado,       --Datos del teclado

     --Senales al RamDac
     hSync => hSync,                -- Sincronizacion horizontal (activa a baja)
     vSync => vSync,               -- Sincronizacion Vertical (activa a baja)
     blanking => blanking,          -- Senyal de blanking (activa a baja)

     --Banco Izquierdo SRam
     WEI => WEI,                          -- Write Enable de la SRAM (activa a baja)
     OEI => OEI,                         -- Output Enable de la SRAM (activa a baja)
     CSI => CSI,                         -- Chip Enable de la SRAM (activa a baja)
     datosMemoriaI => datosMemoriaI,         -- Buffer de datos con la SRAM
     direccionMemoriaI => direccionMemoriaI, -- Direccion de la memoria

     --Banco Derecho SRam
     WED => WED,                          -- Write Enable de la SRAM (activa a baja)
     OED => OED,                         -- Output Enable de la SRAM (activa a baja)
     CSD => CSD,                         -- Chip Enable de la SRAM (activa a baja)
     datosMemoriaD => datosMemoriaD,         -- Buffer de datos con la SRAM
     direccionMemoriaD => direccionMemoriaD, -- Direccion de la memoria

     -- Para iluminar los leds
     estadofsms => estadofsms,

     -- Para pausar las escrituras
     pausa => pausa
    );
--------------------------------------------------------------------------------



--------------------------------------------------------------------------------
-- Maquina de estados del controlador
--------------------------------------------------------------------------------
-- Se encraga de que primero se programe el RamDac y despues empiece a funcionar todo
-- stInicial -> stProgramando -> stFuncionando2
maquinaDeEstados:
process (resetGeneral, relojGeneral)
begin
  if (resetGeneral = '0') then
     estadoActual <= stInicial;
  elsif (relojGeneral'event and relojGeneral='1') then
```

```
    case (estadoActual) is
      when stInicial =>
        estadoActual <= stProgramando;

      when stProgramando => -- Se esta programando el RamDac y espera que acabe
        if (finProgramacionAux = '1') then
          estadoActual <= stFuncionando2;
        else
          estadoActual <= stProgramando;
        end if;

      when stFuncionando =>
        estadoActual <= stFuncionando2;

      when stFuncionando2 => -- Una vez aqui ya permanece siempre
        estadoActual <= stFuncionando2;

      when others =>
        estadoActual <= stInicial;
    end case;

  end if;
end process;
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
-- Salidas del controlador (Moore)
--------------------------------------------------------------------------------
-- Senyales que da la maquina de estados en funcion del estado
senyalesFsm:
process (estadoActual)
begin
  -- Valores por defecto
  resetProgramadorAux <= '1';
  inicioProgramacionAux <= '0';
  resetMaquinaAux <= '1';

  case (estadoActual) is

    when stInicial =>
      resetProgramadorAux <= '0';
      resetMaquinaAux <= '0';

    when stProgramando =>
      inicioProgramacionAux <= '1';
      resetMaquinaAux <= '0';

    when stFuncionando =>
      inicioProgramacionAux <= '0';

    when stFuncionando2 =>
      inicioProgramacionAux <= '0';

  end case;
end process;
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
-- Asignacion concurrente de senyales
--------------------------------------------------------------------------------
  pixelClock <= not(relojGeneral);
  ethernet <= '1';  -- Desactivamos Ethernet para evitar contencion al programar el RamDac
--------------------------------------------------------------------------------


end video_arquitectura;
```

maquina.vhd

```
--------------------------------------------------------------------------------
-- RUTA DE DATOS DEL CONTROLADOR DEL CIRCUITO
--------------------------------------------------------------------------------
-- ENTIDAD: maquina
-- CONEXIONES:
-- PUERTOS ENTRADA:
--    relojMaquina: Reloj de entrada (50 MHz)
--    resetMaquina: Reset general (activo a baja)
-- OTROS PUERTOS (agrupados por funciones):
--    -- Senales de sincronizacion del RamDac
--      hSync (out): Sincronizacion horizontal (activa a baja)
--      vSync (out): Sincronizacion Vertical (activa a baja)
--      blanking (out): Senyal de blanking (activa a baja)
--    -- Banco Izquierdo SRam
--      WEI (out): Write Enable de la SRAM (activa a baja)
--      OEI (out): Output Enable de la SRAM (activa a baja)
--      CSI (out): Chip Enable de la SRAM (activa a baja)
--      datosMemorial[15..0] (buffer): Buffer de datos con la SRAM
--      direccionMemorial[18..0] (out): Direccion de la memoria
--    -- Banco Derecho SRam
--      WED (out): Write Enable de la SRAM (activa a baja)
--      OED (out): Output Enable de la SRAM (activa a baja)
--      CSD (out): Chip Enable de la SRAM (activa a baja)
--      datosMemoriaD[15..0] (buffer): Buffer de datos con la SRAM
--      direccionMemoriaD[18..0] (out): Direccion de la memoria
--    -- Teclado
--      relojTeclado (in): Reloj del teclado
--      datosTeclado (in): Datos del teclado
--------------------------------------------------------------------------------
-- Este modulo implementa la ruta de datos del controlador del circuito
-- Gestiona y controla todo el flujo de datos del circuito en funcion de los datos
--  suministrados, las peticiones del usuario, las senyales internas...
-- Instancia los componentes individuales que son necesarios y los conecta entre si
-- Ademas, incluye algunos componentes sencillos codificados explicitamente
-- La ruta de datos es la que se adjunta en la memoria.
-- La convencion seguida para las senyales es la siguiente:
--  - sXXXX: salida de un componente
--  - cXXXX: senyal del control, carga de registro...
--  - otras: identifican al componente y/o la funcion claramente
--------------------------------------------------------------------------------

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


--------------------------------------------------------------------------------
-- Declaracion de la entidad
--------------------------------------------------------------------------------
entity maquina is
   port (
     -- Vendran de un modulo superior (no son las del sistema)
     resetMaquina: in std_logic;              -- senyal de inicializacion (activa a cero)
     relojMaquina: in std_logic;              -- reloj de entrada

     -- TECLADO
     relojTeclado: IN std_logic;            --Reloj del teclado
     datosTeclado: IN std_logic;            --Datos del teclado

     --Senales al RamDac
     hSync: out std_logic;                -- Sincronizacion horizontal (activa a baja)
     vSync: out std_logic;                -- Sincronizacion Vertical (activa a baja)
     blanking: out std_logic;               -- Senyal de blanking (activa a baja)

     --Banco Izquierdo SRam
     WEI: out std_logic;                -- Write Enable de la SRAM (activa a baja)
     OEI: out std_logic;                -- Output Enable de la SRAM (activa a baja)
     CSI: out std_logic;                -- Chip Enable de la SRAM (activa a baja)
     datosMemorial: inout std_logic_vector(15 downto 0);     -- Buffer de datos con la SRAM
     direccionMemorial: out std_logic_vector(18 downto 0); -- Direccion de la memoria

     --Banco Derecho SRam
     WED: out std_logic;                -- Write Enable de la SRAM (activa a baja)
     OED: out std_logic;                -- Output Enable de la SRAM (activa a baja)
     CSD: out std_logic;                -- Chip Enable de la SRAM (activa a baja)
     datosMemoriaD: inout std_logic_vector (15 downto 0);   -- Buffer de datos con la SRAM
derecho
     direccionMemoriaD: out std_logic_vector (18 downto 0)   -- Direccion de la memoria del banco
derecho

     -- Para iluminar los leds
     estadofsms: out std_logic_vector(4 downto 0);
```

```
    -- Para pausar las escrituras
    pausa: in std_logic;

  );
end maquina;
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
-- Declaracion de la aquitectura
--------------------------------------------------------------------------------
architecture maquina_arquitectura of maquina is


--------------------------------------------------------------------------------
--- Inicio Declaracion de componentes
--------------------------------------------------------------------------------


-------------------------------------------------
--- Para gestionar los ajustes de los efectos
-------------------------------------------------
component Ajustes is
  port (
    relojAjustes: in std_logic;
    vSync: in std_logic;
    resetAjustes: in std_logic;
    tAjuste : in std_logic_vector (1 downto 0);
    tPeticion : in std_logic_vector (3 downto 0);
    dirRefrescoBase: in std_logic_vector (18 downto 0);
    dirRefrescoReg : in std_logic_vector (18 downto 0);
    esZonaWipe: out std_logic;
    esZonaPip: out std_logic;
    frameCounter: out std_logic;
    dirPip: out std_logic_vector( 18 downto 0 );
    dirZoom: out std_logic_vector( 18 downto 0 );
    dirMosaico: out std_logic_vector( 18 downto 0 );
    dirStrobeC: out std_logic_vector( 18 downto 0 );
    dirFlip: out std_logic_vector( 18 downto 0 );
    dirRotacion: out std_logic_vector( 18 downto 0 )
  );
end component;
```

```
-------------------------------------------------
--- El interfaz con la memoria izquierda
-------------------------------------------------
component interfazRam
  port (
    relojInterfaz: in std_logic;
    resetInterfaz: in std_logic;
    hazLectura: in std_logic;
    hazEscritura: in std_logic;
    direccionDeLectura: in std_logic_vector (18 downto 0);
    direccionDeEscritura: in std_logic_vector (18 downto 0);
    datoLeido: out std_logic_vector (15 downto 0);
    datoAEscribir: in std_logic_vector (15 downto 0);
    chipEnable: out std_logic;
    outputEnable: out std_logic;
    writeEnable: out std_logic;
    direccionMemoria: out std_logic_vector (18 downto 0);
    datosMemoria: inout std_logic_vector (15 downto 0)
  );
end component;

-------------------------------------------------
--- Generador de video
-------------------------------------------------
component generadorVideo16b25MHz
  port (
    resetVideo: in std_logic;  -- Reset asincrono (activo a baja)
    relojVideo: in std_logic;  -- Reloj de entrada (funciona con flancos de suubida)
    datosVideo: out std_logic_vector(15 downto 0);    -- Valor RGB 5:5:5 del pixel generado
    direccionVideo: out std_logic_vector(18 downto 0) -- Direccion a la que se corresponde
);
end component;
-------------------------------------------------
--- Generador de senyales
-------------------------------------------------
component generadorSenyales50MHz
  port (
    relojGenerador: in std_logic;  -- Reloj de entrada (funciona con flancos de suubida)
    resetGenerador: in std_logic;  -- Reset asincrono (activo a baja)
    hSync: out std_logic;          -- Sincronizacion horizontal (de linea) (activa a baja)
    vSync: out std_logic;          -- Sincronizacion vertical (de frame) (activa a baja)
    blanking: out std_logic;       -- Senyal de blankin (activa a baja) (1:zona visible;0:fuera)
    pixelCont: out std_logic_vector(10 downto 0); -- Contador de pixeles
    lineaCont: out std_logic_vector(9 downto 0)   -- Contador de lineas
  );
end component;
```

```
-------------------------------------------------
--- Registro Generico
-------------------------------------------------
component RegistroGenerico
   generic (n: integer:=16);
   port(
      clk: in std_logic;
      rst: in std_logic;
      cargar:in std_logic;
      entrada: in std_logic_vector(n-1 downto 0);
      salida : out std_logic_vector(n-1 downto 0));
end component;

-------------------------------------------------
--- Maquinas de estados del controlador
-------------------------------------------------
component fsms
   port (
      relojFsms: in std_logic;
      resetFsms: in std_logic;
      peticion: in std_logic_VECTor (3 downto 0);
      vSync: in std_logic;
      frameCounter: in std_logic;
      esZonaPip: in std_logic;
      esZonaWipe: in std_logic;
      pausa: in std_logic;
     -- Senyales de carga de registros (activos a alta)
      rData: out std_logic;
      sync: out std_logic);
      dadd_in: out std_logic;
      dOE: out std_logic;
      dWR: out std_logic;
      es_efecto_dch: out std_logic;
      -- Senyales de seleccion de multiplexores:
      mux_dadd_lect_efecto: out std_logic; -- 0:Direccion Refresco; 1:Direccion Pip
      mux_dadd_esc_efecto: out std_logic;  -- 0:Direccion Strobe Ciclico; 1:IAdd_In
      mux_dadd_sl: out std_logic;          -- 0:Direccion Lectura(RD); 1:Direccion Escritura(ED)
      mux_iadd_sl: out std_logic_vector(2 downto 0);
      mux_rData_high_sl: out std_logic_VECTor (1 downto 0);
      -- Senyales de apertura de buferes triestado (activos a alta)
      bdData_high_in: out std_logic;
      brData_high: out std_logic;
      -- Senyales que iran al interfaz de la RAM izquierda
      siguienteLeeI: out std_logic;     -- En el siguiente ciclo hara una lectura (activa a alta)
      siguienteEscribeI: out std_logic; -- En el siguiente ciclo hara una escritura (activa a baja)
      -- Para iluminar los leds en funcion del efecto
      estadofsms: out std_logic_vector(4 downto 0)
   );
end component;

-------------------------------------------------
--- Interfaz con el teclado
-------------------------------------------------
component interfaz_teclado_maquina_de_estados
   port (
      reset: in std_logic;
      clock: in std_logic;
      ps2Clk: in std_logic;
      ps2Data: in std_logic;
      vsync : in std_logic;
      registroestado:out std_logic_vector (3 downto 0);
      registroajustes:out std_logic_vector(1 downto 0)
   );
end component;

-------------------------------------------------------------------------------
--- Fin Declaracion de componentes
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
-- Inicio declaración de senyales y constantes
-------------------------------------------------------------------------------


------------------------------
-- Senyales para el modulo de ajustes
------------------------------
   signal sDirRefrescoBase: std_logic_vector (18 downto 0);
   signal sDirRefrescoReg: std_logic_vector (18 downto 0);
   signal sDirPip: std_logic_vector (18 downto 0);
   signal sDirZoom: std_logic_vector (18 downto 0);
   signal sDirMosaico: std_logic_vector (18 downto 0);
   signal sDirStrobeC: std_logic_vector (18 downto 0);
   signal sDirFlip: std_logic_vector (18 downto 0);
   signal sDirRotacion: std_logic_vector (18 downto 0);
   signal sEsZonaWipe: std_logic;
   signal sEsZonaPip: std_logic;
   signal sFrameCounter: std_logic;
------------------------------
```

```
--------------------------------
-- Senyales para conectar el interfaz de la RAM izquierda
--------------------------------
   signal cIAdd_In: std_logic; --Siempre valdr uno
   signal sSiguienteLeeI: std_logic;     -- Desde las fsms
   signal sSiguienteEscribeI: std_logic; -- Desde las fsms
   signal sDatoLeidoI: std_logic_vector(15 downto 0); -- Va a la entrada de RData
   signal sDatosVideo: std_logic_vector(15 downto 0); -- Viene desde el generador de video
   signal sDatosVideoInvertido: std_logic_vector(15 downto 0); -- Viene desde el generador de
video
   signal sDireccionVideo: std_logic_vector(18 downto 0); -- Viene desde el generador de video

--------------------------------
-- Senales para el triple sincronismo
--------------------------------
   signal ssync1: std_logic_vector(2 downto 0);
   signal ssync2: std_logic_vector(2 downto 0);
   signal ssync3: std_logic_vector(2 downto 0);
   signal entradaSync: std_logic_vector(2 downto 0 );

-----------------------
-- senales para las maquinas de estados
-- (solo las salidas, las entradas estaran en los componentes correspondientes)
--------------------------------
      -- Registros a las memorias
   signal cdadd_in: std_logic;
      -- Efectos que acceden a ambas memorias
   signal cEs_Efecto_dch: std_logic;
      -- L/E en memorias
   signal sdoe: std_logic;
   signal sdwr: std_logic;
        --BUFERES
   signal cbddata_high_in: std_logic;
   signal cbrdata_high: std_logic;
        --MUXES
   signal cmux_dadd_esc_efecto: std_logic;
   signal cmux_dadd_lect_efecto: std_logic;
   signal cmux_dadd_sl: std_logic;
   signal cmux_iadd_sl: std_logic_vector(2 downto 0);
   signal cmux_rdata_high_sl: std_logic_vector( 1 downto 0 );
        -- Al RamDac
   signal crdata: std_logic;
   signal csync: std_logic;
--------------------------------
```

```
--------------------------------
-- senales para el teclado
--------------------------------
   signal sPeticion: std_logic_vector(3 downto 0 );
   signal sAjuste: std_logic_vector(1 downto 0 );

--------------------------------
-- senales para el generador de senales
--------------------------------
   signal shSync: std_logic;
   signal svSync: std_logic;
   signal sblanking: std_logic;
   signal slineaCont: std_logic_vector(9 downto 0);
   signal spixelCont: std_logic_vector(10 downto 0);

--------------------------------
-- senales para datos y banco derecho
--------------------------------
   signal sRData: std_logic_vector(15 downto 0);
   signal sMux_RData_High: std_logic_vector(7 downto 0);
   signal sBDData_High_In: std_logic_vector(7 downto 0);
   signal crData_High: std_logic;
   signal sRData_High: std_logic_vector(7 downto 0);
   signal sBRData_High: std_logic_vector(7 downto 0);
   signal sMux_DAdd_Esc:std_logic_vector(18 downto 0);
   signal sMux_DAdd_Lect:std_logic_vector(18 downto 0);
   signal sMux_DAdd:std_logic_vector(18 downto 0);
   signal sDAdd_In:std_logic_vector(18 downto 0);

---------------------------------
-- senales datos y direccion banco izquierdo
   signal sIadd_In: std_logic_vector(18 downto 0);
   signal smux_iadd_in:std_logic_vector(18 downto 0);
   signal smux_iadd_ct:std_logic_vector(1 downto 0);

-------------------------------------------------------------------------------
-- Fin declaracion de constantes y senyales
-------------------------------------------------------------------------------

begin

-------------------------------------------------------------------------------
-- Inicio Codigo
-------------------------------------------------------------------------------
```

```
-------------------------------
-- Modulo de ajustes
-------------------------------
elModuloAjustes: Ajustes
  port map (
    relojAjustes => relojMaquina,
    vSync => sVSync,
    resetAjustes => resetMaquina,
    tAjuste => sAjuste,
    tPeticion => sPeticion,
    dirRefrescoBase => sDirRefrescoBase,
    dirRefrescoReg => sIAdd_In,
    esZonaWipe => sEsZonaWipe,
    esZonaPip => sEsZonaPip,
    frameCounter => sFrameCounter,
    dirPip => sDirPip,
    dirZoom => sDirZoom,
    dirMosaico => sDirMosaico,
    dirStrobeC => sDirStrobeC,
    dirFlip => sDirFlip,
    dirRotacion => sDirRotacion
  );

-------------------------------
-- Interfaz con el banco izquierdo de la memoria
-------------------------------
elInterfazRamI:
interfazRam
  port map (
    relojInterfaz => relojMaquina,
    resetInterfaz => resetMaquina,
    hazLectura => sSiguienteLeeI,
    hazEscritura => sSiguienteEscribeI,
    direccionDeLectura => sMux_IAdd_In,
    direccionDeEscritura => sDireccionVideo,
    datoLeido => sDatoLeidoI,
    datoAEscribir => sDatosVideoInvertido,
    chipEnable => CSI,
    outputEnable => OEI,
    writeEnable => WEI,
    direccionMemoria => direccionMemoriaI,
    datosMemoria => datosMemoriaI
  );
-- Tenemos que invertir la senyal de video
sDatosVideoInvertido <= sDatosVideo(7 downto 0) & sDatosVideo(15 downto 8);
```

```
-------------------------------------------------------------------------------
-- Datos del banco derecho y del ramdac
-------------------------------------------------------------------------------
elrData: RegistroGenerico
generic map(n=>16)
  port map(
    clk => relojMaquina,
    rst => resetMaquina,
    cargar => cRData,      -- Viene de las fsms
    entrada=> sDatoLeidoI, -- Viene del interfazRamI
    salida => sRData       -- Val a Mux_RData_High
);

-- Multiplexor que selecciona que se le pasara al RamDac y la memoria
sMux_RData_High <= sRData(7 downto 0) when cMux_RData_High_sl="00"
                   else sRData(15 downto 8) when cMux_RData_High_sl="01"
                   else sBDData_High_In;
-- Bufer triestado que va al RamDac y la memoria
sBDData_High_In<= sRData(15 downto 8) when cBDData_High_In='1'
                  else "ZZZZZZZZ";

-- Registro que carfa los datos que leemos de la izquierda
crdata_high <= '1';   -- no sale de la fsms, carga siempre
regDataHigh: RegistroGenerico
generic map(n=>8)
  port map(
    clk => relojMaquina,
    rst => resetMaquina,
    cargar => crdata_high,
    entrada => sMux_RData_High,
    salida => sRData_High
);

sBRData_High <= sRData_High when cBRData_High='1'
                else "ZZZZZZZZ";

-------------------------------
-- Bloque direccion banco derecho
-------------------------------
-- Seleccionamos la direccion que usariamos para leer
sMux_DAdd_Lect <= sDirRefrescoBase when cmux_DAdd_Lect_efecto='0'
                  else sDirPip;

-- Seleccionamos la direccion que usariamos para escribir
sMux_DAdd_Esc <= sDirStrobeC when cmux_DAdd_Esc_efecto='0'
                 else sIAdd_In;
```

```
-- Seleccionamos la direccion que vamos a usar
sMux_DAdd <= sMux_DAdd_Lect when cmux_DAdd_Sl='0'
            else sMux_DAdd_Esc;

-- La direccion la cargaremos en un registro
regDAddIn: RegistroGenerico
generic map(n=>19)
   port map(
      clk => relojMaquina,
      rst => resetMaquina,
      cargar => cDAdd_In,
      entrada => sMux_DAdd,
      salida => sDAdd_In
);

------------------------------
-- Bloque Direccion y Datos Banco Izquierdo
------------------------------
-- Seleccion de la direccion de la memoria izquierda
smux_iadd_in <= sdirRefrescoBase when cMux_IAdd_Sl = "000"
                else sdirZoom when cMux_IAdd_Sl = "001"
                else sdirMosaico when cMux_IAdd_Sl = "010"
                else sdirFlip when cMux_IAdd_Sl = "011"
                else sdirRotacion when cMux_IAdd_Sl = "100"
                else sdirRefrescoBase;

-- Registro para la direccion
cIAdd_In <= '1';
elIadd_In:
RegistroGenerico
generic map(n=>19)
   port map(
      clk => relojMaquina,
      rst => resetMaquina,
      cargar => cIAdd_In,
      entrada => sMux_IAdd_In,
      salida => sIAdd_In
);

-- Direccion base compuesta por el generador de senyales
sDirRefrescoBase <= slineaCont(8 downto 0) & spixelcont(10 downto 1);
```

```
------------------------------
-- Bloque para el pipeline de las senyales de sincronismo
------------------------------
-- Encadenamos tres registros para adecuar el pipeline
entradaSync <= shSync & svSync & sblanking;
regSync1: RegistroGenerico
generic map(n=>3)
   port map(
      clk => relojMaquina,
      rst => resetMaquina,
      cargar => csync,
      entrada => entradaSync,
      salida => ssync1
);

regSync2: RegistroGenerico
generic map(n=>3)
   port map(
      clk => relojMaquina,
      rst => resetMaquina,
      cargar => csync,
      entrada => ssync1,
      salida => ssync2
);

regSync3: RegistroGenerico
generic map(n=>3)
   port map(
      clk => relojMaquina,
      rst => resetMaquina,
      cargar => csync,
      entrada => ssync2,
      salida => ssync3
);

------------------------------
-- Comunicaciones con el generador de video
------------------------------
elgeneradorVideo: generadorVideo16b25MHz
   port map (
      resetVideo => resetMaquina,
      relojVideo => relojMaquina,
      datosVideo => sDatosVideo,
      direccionVideo => sDireccionVideo
);
```

```
-------------------------------
-- Instanciacion del interfaz del teclado
-------------------------------
elInterfazTeclado: interfaz_teclado_maquina_de_estados
  port map (
    reset => resetMaquina,
    clock => relojMaquina,
    ps2Clk => relojTeclado,
    ps2Data => datosTeclado,
    vsync => ssync3(1),
    registroEstado => sPeticion,
    registroAjustes => sAjuste
  );


-------------------------------
-- Generacion de las senyales
-------------------------------
elgeneradorSenales: generadorSenyales50MHz
  port map (
    relojGenerador => relojMaquina,
    resetGenerador => resetMaquina,
    hSync => shSync,
    vSync => svSync,
    blanking => sBlanking,
    pixelCont => sPixelCont,
    lineaCont => sLineaCont
);

-------------------------------
-- Instanciacion de las maquinas de estados
-------------------------------
lasFsms: fsms
port map(
      relojFsms => relojMaquina,
      resetFsms => resetMaquina,
      esZonaPip => sEsZonaPip,
      esZonaWipe => sEsZonaWipe,
      es_efecto_dch => ces_Efecto_dch,
      frameCounter => sFrameCounter,
      peticion => sPeticion,
      vSync => sVSync,
      -- Resistros a las memorias
      dadd_in =>  cdadd_in,
      -- L/E en memorias
      doe => sdoe,
      dwr => sdwr,
      --BUFERES
      bddata_high_in => cbddata_high_in,
      brdata_high => cbrdata_high,
      --MUXES
      mux_dadd_esc_efecto => cmux_dadd_esc_efecto,
      mux_dadd_lect_efecto => cmux_dadd_lect_efecto,
      mux_dadd_sl => cmux_dadd_sl,
      mux_iadd_sl => cmux_iadd_sl,
      mux_rdata_high_sl => cmux_rdata_high_sl,
      -- Al RamDac
      rdata => crdata,
      sync => csync
      siguienteLeeI => sSiguienteLeeI,
      siguienteEscribeI => sSiguienteEscribeI,
      estadofsms => estadofsms,
      pausa => pausa,
);

-------------------------------
-- ASIGNACION DE SENYALES DE SALIDA DEL MODULO
-- "Todas" salen desde senyales internas
-------------------------------
  -- Al RamDac
  hSync <= ssync3(2);
  vSync <= ssync3(1);
  blanking <= ssync3(0);

  -- Banco Derecho
  WED <= sdwr; --Directas desde la fsms
  OED <= sdoe; --Directas desde la fsms
  CSD <= '0';
  datosMemoriaD <= sBDData_High_In & sBRData_High;
  direccionMemoriaD <= sDAdd_In;


end maquina_arquitectura;
```

generadorvideo16b25mhz.vhd

```
--------------------------------------------------------------------------------
-- PSEUDO-GENERADOR DE VIDEO CON RESOLUCION DE 628 x 480 Y MUESTRAS
-- DE 16 BITS EN FORMATO RGB 5:5:5 Y A FRECUENCIA DE 25MHz
--------------------------------------------------------------------------------
-- ENTIDAD: generadorVideo16b25MHz
-- CONEXIONES:
-- PUERTOS ENTRADA:
--    relojVideo: Entrada del reloj a 50MHz (funciona con flnaco de subida)
--    resetVideo: Reset asincrono del circuito (activo a baja)
-- PUERTOS SALIDA:
--    datosVideo[15..0]: Valor RGB 5:5:5 del pixel generado
--    direccionVideo[18..0]: Direccion a la que se corresponde el pixel generado
--------------------------------------------------------------------------------
-- Este modulo simula el comportamiento de un CoDec de video.
-- Genera muestras de 16 bits con formato RGB 5:5:5, a una frecuencia de 25Mhz
--  (una nueva cada dos ciclos del reloj de entrada). Ademas va generando la
--  la direccion de memoria a la que se corresponderia dicho pixel.
-- El patron que genera es el de un cuadrado de 150x150 pixeles que se desplaza
--  desde la esquina superior-izquierda hacia la inferior-derecha. Avanza un pixel
--  por cada 8 frames refrescados (para evita que vaya demasiado rapido). Como
--  fondo se ha colocado un degradado cromatico que nos permite comprobar la
--  correcta generacion de los colores. Ademas, se dibuja un recuadro blanco a modo
--  de borde con lineas de 15 pixeles de grosor y separadas de los margenes 5
--  pixeles para comprobar que tenemos correctamente situados los ejes de coordenadas
--  y que todo esta corerctamente sincronizado.
-- Puesto que el objetivo es simular dicho comportamiento para poder comprobar
--  el funcionamiento del resto de componentes, lo hemos adaptado en la medida
--  de lo posible para que facilite la interaccion con el resto.
--
-- De esta forma, el comportamiento del modulo seria:
-- - Genera muestras de 16 bits con formato RGB 5:5:5. Un CoDec las generaria de
--   8 bits pero, haciendolo de esta manera, evitamos el tener que "componer" la
--   muestra total con cada dos que recibamos.
-- - Genera una nueva muestra a una frecuencia de 25MHz. Un CoDec tambien funciona
--   a esta frecuencia, solo que sigue un patron Cb, Y, Cr, Y, Cb, Y,... y nosotros
--   estamos generando siempre muestras con informacion "valida".
-- - El modulo entrega ya la direccion de memoria a la que se corresponde el pixel
--   que se acaba de generar. Esto no lo hace un CoDec, pero lo hemos incluido por
--   dos motivos:
--    - De momento evitamos tener que hacerlo en el controlador, que tendria que
--      llevar una cuenta de los pixeles que ha ido recibiendo y almacenando.
--    - Evitamos la sincronizacion explicita entre el CoDec y el controlador, con
--      lo que podemos olvidarnos de todas las senyales de inicio/fin de linea,
--      inicio/fin de frame,... y lo unico que tenemos que hacer es recoger lo que
--      nos entregue el generador (datos y direccion) y escribirlo cuando toque, pues
--      es el propio generador quien se encarga de controlar lo que va generando.
--
-- Otras consideraciones:
-- - Para conseguir una buena sincronizacion y evitar problemas potenciales, se
--   ha partido del modulo que genar todas las senyales de sincronismo para el
--   resto del circuito.
-- - Aunque no seria necesario porque deberia hacerl el controlador, las senyales
--   se sacan latcheadas mediante un registro que tiene como senyal de enable
--   el reloj de 25MHz.
-- - Se han dejado todas las senyales internas que se utilizan a la hora de generar
--   estas senyales, principalmente por si las queremos usar a la hora de simular
--   el comportamiento del CoDec.
--   El funcionamiento de estas senyales, puede verse en el siguiente grafico:
--      |<---- Region Activa ---->|<---------- Region de Blanking --------->|
--      |                        |                                         |
--   ----+--------- ... ----------+---------------          --------------+---
--   |  |                        |           |           |           |
--   |  |                        |<--Porche    |<-- Tiempo   |<-- Porche   |
--   |  |                        |  Delantero-->|    Sinc--->|  Trasero -->|
-- ---  |                        |           ----------          |
--      |                        |                                         |
--      |<----------------------- Periodo ----------------------------------->|
--
-- Nota:
-- - Puesto que estamos generando las direcciones sobre las que se va a escribir,
--   hay que tener en cuenta el rango en el que nos estamos moviendo para que el
--   controlador genere las de refresco dentro del mismo rango y evitar estar
--   refrescando direcciones sobre las que nunca escribiremos o que estemos escribiendo
--   sobre direcciones que se corresponden con otras posiciones por haberle dado
--   la "vuelta" al contador correspondiente.
--------------------------------------------------------------------------------


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

```
----------------------------------------------------------------------------
-- Declaracion de la entidad
----------------------------------------------------------------------------
entity generadorVideo16b25MHz is
  port (
    resetVideo: in std_logic;  -- Reset asincrono (activo a baja)
    relojVideo: in std_logic;  -- Reloj de entrada (funciona con flancos de suubida)
    datosVideo: out std_logic_vector(15 downto 0);    -- Valor RGB 5:5:5 del pixel generado
    direccionVideo: out std_logic_vector(18 downto 0) -- Direccion a la que se corresponde el
pixel generado
  );
end generadorVideo16b25MHz;
----------------------------------------------------------------------------


----------------------------------------------------------------------------
-- Declaracion de la arquitectura
----------------------------------------------------------------------------
architecture generadorVideo16b25MHz_Arquitectura of generadorVideo16b25MHz is


-- Constantes para la temporizacion de las senyales:
  -- Temporizacion de linea (pixeles):
  constant H_PIXELES: INTEGER:= 1256;   -- 1256(=628x2) porque funcionamos a 50MHz)
  constant H_PORCHEDEL:   INTEGER:= 48; -- Porche delantero
  constant H_SINC: INTEGER:= 189;       -- Tiempo de sincronizacion
  constant H_PORCHETRAS: INTEGER:= 95;  -- Porche trasero
  constant H_PERIODO: INTEGER:= H_SINC + H_PIXELES + H_PORCHEDEL +
H_PORCHETRAS; -- Periodo de linea

  -- Temporizacion de frame (lineas):
  constant V_LINEAS: INTEGER:= 480;     -- 480 lineas (independientemente del numero de
pixeles)
  constant V_PORCHEDEL: INTEGER:= 14;   -- Porche delantero
  constant V_SINC: INTEGER:= 2;         -- Tiempo de sincronizacion
  constant V_PORCHETRAS: INTEGER:= 32;  -- Porche trasero
  constant V_PERIODO: INTEGER:= V_SINC + V_LINEAS + V_PORCHEDEL +
V_PORCHETRAS; --Periodo de frame

-- Contadores
  signal pixelCont: std_logic_vector(10 downto 0);   --Contador de pixeles (horizontal)
  signal lineaCont: std_logic_vector(9 downto 0);    --Contador de lineas (vertical)

-- Senyales auxiliares (hacen falta porque no podemos leer de los puertos de salida)
  signal hSyncAux: std_logic;
  signal vSyncAux: std_logic;
  signal blankingAux: std_logic;

--Relojes axiliares (por si nos hacen falta)
  signal reloj25MHz: std_logic;   -- Reloj auxiliar a 25Mhz
  signal reloj125MHz: std_logic;  -- Reloj auxiliar a 12'5Mhz

--Senyales para simular los patrones
  signal offsetColor : std_logic_vector(15 downto 0); -- Desplazamiento del color
  signal offsetX : std_logic_vector(9 downto 0);      -- Desplazamiento en el eje X
  signal offsetY : std_logic_vector(8 downto 0);      -- Desplazamiento en el eje Y
  signal dentroCuadrado: std_logic;                   -- 1=Dentro del cuadrado; 0=fuera
  signal bordeCuadro: std_logic;                      -- 1 Cuando sea un borde; 0 e.o.c.
  signal contadorFrames: std_logic_vector(2 downto 0);
  signal patronVideo: std_logic_vector(15 downto 0);  -- Senyal auxiliar que luego se sacara por
la salida de datos
  signal direccionVideoAux: std_logic_vector(18 downto 0);

-- Pruebas de patrones en color
  constant KColor_31: std_logic_vector(4 downto 0):= "11111";
  signal ascendente: std_logic_vector(4 downto 0);
  signal descendente: std_logic_vector(4 downto 0);
  signal modoVariacionComponente: std_logic_vector(2 downto 0);
  signal degradadoColor: std_logic_vector(15 downto 0);
  signal cambiaModo: std_logic;

begin

----------------------------------------
-- Relojes internos auxiliares a 25 y 12'5MHz
----------------------------------------
reloj25MHz <= pixelCont(0);
reloj125MHz <= pixelCont(1);
----------------------------------------


----------------------------------------
-- Contador de pixeles
----------------------------------------
-- Es un contador modulo H_PERIODO que se incrementa con cada flanco de
--  subida del reloj de entrada.
----------------------------------------
ContadorPixeles:
process(relojVideo, resetVideo)
begin
  -- El resetVideo a cero inicializa el valor del contador
  if (resetVideo = '0') then
    pixelCont <= (others => '0');
  -- Solo responde ante flanco de subida del reloj
  elsif (relojVideo'event and relojVideo = '1') then
```

```
    -- Se incrementara hasta llegar al valor del periodo horizontal (num. pixeles)
    if (pixelCont < H_PERIODO) then
      pixelCont <= pixelCont + 1;
    else
      pixelCont <= (others => '0');
    end if;
  end if;
end process;
-----------------------------------------



-----------------------------------------
-- Contador de lineas
-----------------------------------------
-- Es un contador modulo H_LINEAS que se incrementa con cada flanco de
--  subida de la senyal de sincronizacion horizontal (cada vez que hemos
--  terminado de mostrar una linea).
-----------------------------------------
ContadorLineas:
process(hSyncAux, resetVideo)
begin
  -- El resetVideo a cero inicializa el valor del contador
  if (resetVideo='0') then
    lineaCont <= (others => '0');
  -- Solo responde ante flanco de subida de la senyal de sincronizacion horizontal
  elsif (hSyncAux'event and hSyncAux = '1') then
    -- Se incrementara hasta llegar al valor del periodo vertical (num. lineas)
    if (lineaCont < V_PERIODO) then
      lineaCont <= lineaCont + 1;
    else
      lineaCont <= (others => '0');
    end if;
  end if;
end process;
-----------------------------------------



-----------------------------------------
-- Implementacion de la senyal de sincronizacion horizontal (de linea)
-----------------------------------------
-- Depende del valor del contador de pixels y se activara cuando el contador
--  haya superado el valor del porche delantero mas el numero de pixeles visibles.
--------------------
SincronizaciónHorizontal:
process(relojVideo, resetVideo)
begin
  -- El resetVideo a cero la desactiva (la pone a uno)
  if (resetVideo = '0') then
    hSyncAux <= '1';
  -- Responde ante los flancos de subida del reloj
  elsif (relojVideo'event and relojVideo = '1') then
    -- Dependiendo del intervalo, se activa para sincronizar o no e indicar una nueva linea
    if (pixelCont >= (H_PORCHEDEL + H_PIXELES) and pixelCont < (H_PIXELES +
H_PORCHEDEL + H_SINC)) then
      hSyncAux <= '0';
    else
      hSyncAux <= '1';
    end if;
  end if;
end process;
-----------------------------------------



-----------------------------------------
-- Implementacion de la senyal de sincronizacion vertical (de frame)
-----------------------------------------
-- Depende del valor del contador de linea y se activara cuando el contador
--  haya superado el valor del porche delantero mas el numero de lineas visibles.
-----------------------------------------
SincronizacionVertical:
process(hSyncAux, resetVideo)
begin
  -- El resetVideo a cero la desactiva (la pone a uno)
  if (resetVideo = '0') then
    vSyncAux <= '1';
  -- Responde ante los flancos de subida de la senyal de sincronizacion horizontal
  elsif (hsyncAux'event and hSyncAux = '1') then
    -- Dependiendo del intervalo, se activa para sincronizar o no e indicar un nuevo frame
    if (lineaCont >= (V_LINEAS + V_PORCHEDEL) and lineaCont < (V_LINEAS +
V_PORCHEDEL + V_SINC)) then
      vSyncAux <= '0';
    else
      vSyncAux <= '1';
    end if;
  end if;
end process;
-----------------------------------------
```

```
-----------------------------------------
-- Implementacion de la senyal de blanking
-----------------------------------------
-- Depende del valor del contador de pixels y del de lineas.
-- Activa a cero. Cuando estemos fuera de la zona visible de la pantalla, hay
--  que hacer blanking, por lo que la ponemos a cero. En zona visible, vale uno.
-----------------------------------------
senyalDeBlanking:
process (relojVideo, resetVideo)
begin
  if (resetVideo = '0') then
    blankingAux <= '0';
  elsif (relojVideo'EVENT and relojVideo = '1') then
    -- Cero cuando estamos fuera y uno cuando estamos dentro
    if (pixelCont >= H_PIXELES or lineaCont >= V_LINEAS) then
      blankingAux <= '0';
     else
       blankingAux <= '1';
     end if;
  end if;
end process;
-----------------------------------------


-----------------------------------------
-- Calculo de los desplazamientos que luego usaremos en los patrones
-- En cada frame, modifica los valores de los desplazamientos
-----------------------------------------
process(resetVideo, vSyncAux)
begin
  if (resetVideo = '0') then
    offsetColor <= (others => '0');
    offsetX <= (others => '0');
    offsetY <= (others => '0');
    contadorFrames <= (others => '0');
  elsif (vSyncAux'event and vSyncAux = '1') then
    offsetColor <= offsetColor + 1;
    if (contadorFrames = "011") then
      offsetX <= offsetX + 2;
      offsetY <= offsetY + 1;
      contadorFrames <= "000";
    else
      contadorFrames <= contadorFrames + 1;
    end if;
  end if;
end process;
```

```
-----------------------------------------
-- Vale uno cuando estamos dentro del cuadrado que estamos dibujando o cero fuera de el
-----------------------------------------
dentroCuadrado <= '1' when (pixelCont>offsetX and pixelCont<offsetX+150 and
lineaCont>offsetY and lineaCont<offsetY+75)
                 else '0';
-----------------------------------------


-----------------------------------------
-- Vale uno cuando estamos en alguno de los bordes del cuadro que dibujamos como margen
-----------------------------------------
bordeCuadro <= '1' when (pixelCont>10 and pixelCont<40) or (pixelCont>1216 and
pixelCont<1246)
              or (lineaCont>5 and lineaCont<20) or (lineaCont>460 and lineaCont<475)
              else '0';
-----------------------------------------


-----------------------------------------
-- Direccion a la que se corresponde el bit que estamos generando
-----------------------------------------
direccionVideoAux <= lineaCont(8 downto 0) & pixelCont(10 downto 1);
-----------------------------------------


-----------------------------------------
-- Sacamos los datos y la direccion latcheados con el reloj de 25MHz
-----------------------------------------
-- Se pasan por un biestable en el que la senyal de enable es el reloj de 25MHz
-----------------------------------------
process (resetVideo, relojVideo)
begin
  if (resetVideo = '0') then
    datosVideo <= (others => '0');
    direccionVideo <= (others => '0');
  elsif (relojVideo'event and relojVideo='1') then
    if (reloj25MHz = '1') then
      datosVideo <= patronVideo;
      direccionVideo <= direccionVideoAux;
    end if;
  end if;
end process;
-----------------------------------------
```

```
---------------------------------------
-- Generacion del patron de video que mostraremos
---------------------------------------
-- Calculo del valor de la componente de color base
  ascendente <= pixelCont(6 downto 2);
  descendente <= not(pixelCont(6 downto 2));

-- Bit que nos indica que tenemos que saltar al siguiente modo
  cambiaModo <= pixelCont(7);

-- Calculo del siguiente modo de color
process (resetVideo, cambiaModo)
begin
  if (resetVideo = '0') then
     modoVariacionComponente <= "000";
  elsif (cambiaModo'event and cambiaModo='1') then
    if (modoVariacionComponente = 5) then
      modoVariacionComponente <= "000";
    else
      modoVariacionComponente <= modoVariacionComponente + 1;
    end if;
  end if;
end process;

-- Obtenemos el color que toque
degradadoColor <=      "0" & "11111"     & ascendente  & "00000"      when
modoVariacionComponente="000"
               else "0" & descendente & "11111"     & "00000"      when
modoVariacionComponente="001"
               else "0" & "00000"     & "11111"     & ascendente   when
modoVariacionComponente="010"
               else "0" & "00000"     & descendente & "11111"      when
modoVariacionComponente="011"
               else "0" & ascendente  & "00000"     & "11111"      when
modoVariacionComponente="100"
               else "0" & "11111"     & "00000"     & descendente  when
modoVariacionComponente="101"
               else "0" & "11111"     & "00000"     & descendente;  -- El mismo que el "101"
---------------------------------------
```

```
---------------------------------------
-- Seleccion del patron que corresponda en funcion de la zona que estemos refrescando
---------------------------------------
patronVideo <= (others=>'0') when (dentroCuadrado='1' and bordeCuadro='0')
          else (others=>'1') when (dentroCuadrado='0' and bordeCuadro='1')
          else (others=>'1') when (dentroCuadrado='1' and bordeCuadro='1')
          else degradadoColor;-- when (dentroCuadrado='0' and bordeCuadro='0')
---------------------------------------

end generadorVideo16b25MHz_Arquitectura;
```

fsms.vhd

```
--------------------------------------------------------------------------------
-- MAQUINAS DE ESTADOS (CICLOS Y EFECTOS) DEL CONTROLADor
--------------------------------------------------------------------------------
-- ENTIDAD: fsms
-- CONEXIONES:
-- PUERTOS ENTRADA:
--   relojFsms: Reloj General del sistema (a 50MHz)
--   resetFsms: Reset General del sistema (activo a baja)
--   vSync: Senyal de sincronización vertical
--   peticion(3..0): Codificacion de la peticion que ha hecho el usuario
--   frameCounter: fin de cuenta del contador de frames (activo a alta)
--   esZonaPip: Indica si estamos dentro del cuadro o no (activo a alta)
--   esZonaWipe: Indica si estamos en la zona estatica o no (activo a alta)
--   pausa: Para pausar las escrituras manualmente
-- PUERTOS SALIDA:
--   -- Senyales de carga de registros (activos a alta)
--   dadd_in: Registro de la direccion del banco derecho
--   rData: registro de los datos leidos de la memoria izquierda
--   sync: bloque de tres registros de las senyales de sincronizacion
--   dOE: senyal de lectura del banco derecho
--   dWR: senyal de escritura del banco derecho
--   es_efecto_dch: senyales que afectan a efectos del banco derecho
--   -- Senyales de seleccion de multiplexores:
--   mux_dadd_lect_efecto: Direccion de lectura del banco derecho
--   mux_dadd_esc_efecto: Direccion de escritura del banco derecho
--   mux_dadd_sl: Direccion con la que se atacara el banco derecho
--   mux_iadd_sl(1..0): Direccion con la que se atacara el banco izquierdo
--   mux_rData_high_sl(1..): Dato que se le pasara al RamDac
--   -- Senyales de apertura de buferes triestado (activos a alta)
--   bdData_high_in: Bufer bidireccional con la parte alta de la memoria derecha
--   brData_high_in: Bufer bidireccional con la parte baja de la memoria derecha
--   -- Senyales que iran al interfaz de la RAM izquierda
--   siguienteLeel: hace una peticion de lectura (activo a alta)
--   siguienteEscribel: hace una peticion de lectura (activo a alta)
--   -- Para iluminar los leds en funcion del efecto
--   estadofsms(4..0)
--------------------------------------------------------------------------------
-- En este modulo implementamos las dos maquinas de estados que gestionan el
--  comportamiento general del circuito.
-- La de ciclos es la encargada de la alternancia estricta entre los diferentes
--  modos de acceso a las memorias y la generacion de las senyales en funcion
--  de lo que estemos haciendo: escribiendo en la izquierda (EI), en la
--  derecha (ED), refrescando de la izquierda (RI) o de la derecha (RD).
-- La de efectos se encarga de controlar a uqe efecto hay que pasar en funcion
--  de la peticion que haya hecho el usuario o de la zona en la que nos encontremos.
--------------------------------------------------------------------------------


library IEEE;
use IEEE.std_logic_1164.all;

use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

library SYNOPSYS;
use SYNOPSYS.attributes.all;


--------------------------------------------------------------------------------
-- Declaracion de la entidad
--------------------------------------------------------------------------------
entity fsms is
  port (
    relojFsms: in std_logic;
    resetFsms: in std_logic;
    peticion: in std_logic_VECTor (3 downto 0);
    vSync: in std_logic;
    frameCounter: in std_logic;
    esZonaPip: in std_logic;
    esZonaWipe: in std_logic;
    pausa: in std_logic;
   -- Senyales de carga de registros (activos a alta)
    rData: out std_logic;
    sync: out std_logic);
    dadd_in: out std_logic;
    dOE: out std_logic;
    dWR: out std_logic;
    es_efecto_dch: out std_logic;
    -- Senyales de seleccion de multiplexores:
    mux_dadd_lect_efecto: out std_logic; -- 0:Direccion Refresco; 1:Direccion Pip
    mux_dadd_esc_efecto: out std_logic;  -- 0:Direccion Strobe Ciclico; 1:IAdd_In
    mux_dadd_sl: out std_logic;          -- 0:Direccion Lectura(RD); 1:Direccion Escritura(ED)
    mux_iadd_sl: out std_logic_vector(2 downto 0);
    mux_rData_high_sl: out std_logic_VECTor (1 downto 0);
```

```
    -- Senyales de apertura de buferes triestado (activos a alta)
    bdData_high_in: out std_logic;
    brData_high: out std_logic;
    -- Senyales que iran al interfaz de la RAM izquierda
    siguienteLeeI: out std_logic;     -- En el siguiente ciclo hara una lectura (activa a alta)
    siguienteEscribeI: out std_logic; -- En el siguiente ciclo hara una escritura (activa a baja)
    -- Para iluminar los leds en funcion del efecto
    estadofsms: out std_logic_vector(4 downto 0)
end;
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
-- Declaracion de la arquitectura
--------------------------------------------------------------------------------
architecture fsms_Arquitectura of fsms is


-- Estados de la maquina de ciclos
  type tCiclos is (stInicial, eD, eI, rD, rI);
  signal ciclos: tCiclos;

-- Estados de la maquina de efectos:
  type tEfectos is (mosaico, normal, pip, recall, still1f, stillRf, wipe, zum,
                    strobe1f, strobeRf, strobeC1f, strobeCRf, flip, rotate);
  signal efectos: tEfectos

-- Constantes para codificar los efectos
  constant tCANCEL  : std_logic_VECTor (3 downto 0) := "0000";
  constant tSTILL   : std_logic_VECTor (3 downto 0) := "0001";
  constant tRECALL  : std_logic_VECTor (3 downto 0) := "0010";
  constant tSTROBE  : std_logic_VECTor (3 downto 0) := "0011";
  constant tWIPE    : std_logic_VECTor (3 downto 0) := "0100";
  constant tPIP     : std_logic_VECTor (3 downto 0) := "0101";
  constant tZUM     : std_logic_VECTor (3 downto 0) := "0110";
  constant tMOSAICO : std_logic_VECTor (3 downto 0) := "0111";
  constant tFLIP    : std_logic_VECTor (3 downto 0) := "1000";
  constant tROTATE  : std_logic_VECTor (3 downto 0) := "1001";
  constant tSTROBEC : std_logic_VECTor (3 downto 0) := "1010";


begin
```

```
--------------------------------------------------------------------------------
-- Generamos el siguiente ciclo y las salidas tipo Mealy
--------------------------------------------------------------------------------
generacionDelSiguienteCiclo:
process (relojFsms, resetfsms, efectos, esZonaWipe, esZonaPip)
begin
  if (resetFsms = '0') then
    ciclos <= stInicial;
  elsif (relojFsms'event and relojFsms = '1') then

    case ciclos is

      when stInicial =>
        if (efectos = stillrf) or (efectos = recall) or
           (efectos = strobeRf) or (efectos = strobeCRf) or
           (efectos = wipe and esZonaWipe='1') or (efectos = pip and esZonaPip='1') then
             ciclos <= rD;
        elsif (efectos = still1f) or (efectos = strobe1f) or (efectos = strobeC1f) then
             ciclos <= eD;
        elsif (efectos = normal) or (efectos = zum) or (efectos = mosaico) or
             (efectos = rotate) or (efectos = flip) or
              (efectos = wipe and esZonaWipe='0') or (efectos = pip and esZonaPip='0') then
             ciclos <= rI;
        else
           ciclos <= rI;
        end if;

      when eD => Escribe Derecha
        -- Siempre iremos a un Escribe Izquierda
        ciclos <= eI;

      when rD => -- Refresco Derecha
        -- Siempre iremos a un Escribe Izquierda
       ciclos <= eI;

      when rI => -- Refresco Izquierda
        -- Siempre iremos a un Escribe Izquierda
        ciclos <= eI;
```

```
        when eI => -- Escribe Izquierda
          -- El salto depende del efecto en el que nos encontremos
          if (efectos = stillrf) or (efectos = recall) or
             (efectos = strobeRf) or (efectos = strobeCRf) or
             (efectos = wipe and esZonaWipe='1') or (efectos = pip and esZonaPip='1') then
               ciclos <= rD;
          elsif (efectos = still1f) or (efectos = strobe1f) or (efectos = strobeC1f) then
               ciclos <= eD;
          elsif (efectos = normal) or (efectos = zum) or (efectos = mosaico) or
               (efectos = rotate) or (efectos = flip) or
                 (efectos = wipe and esZonaWipe='0') or (efectos = pip and esZonaPip='0') then
               ciclos <= rI;
          else
             ciclos <= rI;
          end if;

      end case;
    end if;
end process;
-------------------------------------------------------------------------------



-------------------------------------------------------------------------------
-- Generacion de las senyales de salida en funcion del estado
-------------------------------------------------------------------------------
maquinaDeCiclos:
process (ciclos, efectos, pausa, esZonaWipe, esZonaPip)

begin

   -- Valores por defecto
   siguienteLeeI <= '0';
   siguienteEscribeI <= '0';
   mux_iadd_sl <= "000";
   es_efecto_dch <= '0';
   dadd_in<='0';
   mux_dadd_sl<='0';
   mux_dadd_lect_efecto<='0';
   mux_dadd_esc_efecto<='0';
   bdData_high_in<='0';
   mux_rData_high_sl<="00";
   brData_high<='1';
   dWR<='1';
   dOE<='1';
   rData<='0';
   sync<='0';

-- Seleccionamos las senyales en funcion del tipo de ciclo
case (ciclos) is

   -- Estado inicial
   when stInicial =>
      siguienteLeeI <= '0';
      siguienteEscribeI <= '0';
      mux_iadd_sl<="000";
      es_efecto_dch<='0';
      dadd_in<='0';
      mux_dadd_sl<='0';
      mux_dadd_lect_efecto<='0';
      mux_dadd_esc_efecto<='0';
      bdData_high_in<='0';
      mux_rData_high_sl<="00";
      brData_high<='1';
      dWR<='1';
      dOE<='1';
      rData<='0';
      sync<='0';

   -- Escritura en el banco derecho
   when eD =>
      siguienteLeeI <= '0';
      siguienteEscribeI <= pausa; --'1';
      mux_iadd_sl<="000";
      es_efecto_dch<='0';
      dadd_in<='1';
      mux_dadd_sl<='1';          -- Sale una direccion de escritura
      mux_dadd_lect_efecto<='0';
      case (efectos) is  -- Diferenciamos strobeC de still y strobe
        when strobeC1f => mux_dadd_esc_efecto <='0';
        when others    => mux_dadd_esc_efecto <='1';
      end case;
      bdData_high_in<='1';
      mux_rData_high_sl<="01";
      brData_high<='1';
      dWR<='0';
      dOE<='1';
      rData<='1';
      sync<='1';
```

```
-- Escritura en el banco izquierdo
when eI =>
   if (efectos = stillrf) or (efectos = recall) or
      (efectos = strobeRf) or (efectos = strobeCRf) or
      (efectos = wipe and esZonaWipe='1') or (efectos = pip and esZonaPip='1') then
         -- Vamos a ir a un RD
         siguienteLeeI <= '0';
         siguienteEscribeI <= '0';
   elsif (efectos = still1f) or (efectos = strobe1f) or (efectos = strobeC1f) then
         -- Vamos a ir a un ED
         siguienteLeeI <= '1';
         siguienteEscribeI <= '0';
   elsif (efectos = normal) or (efectos = zum) or (efectos = mosaico) or
         (efectos = rotate) or (efectos = flip) or
          (efectos = wipe and esZonaWipe='0') or (efectos = pip and esZonaPip='0') then
         -- Vamos a ir a un RI
         siguienteLeeI <= '1';
         siguienteEscribeI <= '0';
   else
      -- Por defecto, aun RI
      siguienteLeeI <= '0';
      siguienteEscribeI <= '0';
   end if;
   case (efectos) is -- Seleccion de la direccion de la memoria izquierda
      when normal  => Mux_IAdd_Sl <= "000";
      when zum     => Mux_IAdd_Sl <= "001";
      when mosaico => Mux_IAdd_Sl <= "010";
      when flip    => Mux_IAdd_Sl <= "011";
      when rotate  => Mux_IAdd_Sl <= "100";
      when others  => Mux_IAdd_Sl <= "000";
   end case;
   es_efecto_dch<='0';
   dadd_in<='0';
   mux_dadd_sl<='0';
   mux_dadd_lect_efecto<='0';
   mux_dadd_esc_efecto<='0';
   bdData_high_in<='0';
   mux_rData_high_sl<="00";
   brData_high<='1';
   dWR<='1';
   dOE<='1';
   rData<='0';
   sync<='0';

      -- Refresco del banco derecho
      when rD =>
         siguienteLeeI <= '0';
         siguienteEscribeI <= pausa;-- '1';  -- Despues escribiremos si no pulsado (pausa=1)
         mux_iadd_sl<="000";
         es_efecto_dch<='0';
         dadd_in<='1';
         mux_dadd_sl<='0';          -- Sale una direccion de lectura
         case (efectos) is          -- Diferenciamos pip de recall, wipe,...
            when pip    => mux_dadd_lect_efecto <= '1';
            when others => mux_dadd_lect_efecto <= '0';
         end case;
         mux_dadd_esc_efecto<='0';
         bdData_high_in<='0';
         mux_rData_high_sl<="10";
         brData_high<='0';
         dWR<='1';
         dOE<='0';
         rData<='0';
         sync<='1';

      -- Refresco del banco izquierdo
      when rI =>
         siguienteLeeI <= '0';
         siguienteEscribeI <= pausa;-- '1';  -- Despues escribiremos si no pulsado (pausa=1)
         mux_iadd_sl <= "000";
         es_efecto_dch <= '1';
         dadd_in <= '1';
         mux_dadd_sl<='0';
         mux_dadd_lect_efecto<='0';
         mux_dadd_esc_efecto<='0';
         bdData_high_in<='0';
         mux_rData_high_sl <="01";
         brData_high<='1';
         dWR<='1';
         dOE<='1';
         rData<='1';
         sync<='1';

   end case;

end process;
```

```
--------------------------------------------------------------------------------
-- Generacion del siguiente efecto (es una fsm con vSync como reloj)
--------------------------------------------------------------------------------
maquinaDeEfectos:
process (vSync, resetfsms)

begin

  if (resetFsms = '0') then
    efectos <= normal;

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

    case (efectos) is

      when mosaico =>
        if peticion=tCANCEL then efectos <= normal;
        else efectos <= mosaico;
        end if;

      when normal =>
        case (peticion) is
          when tRECALL  => efectos <= recall;
          when tSTROBE  => efectos <= strobe1f;
          when tSTROBEC => efectos <= strobeC1f;
          when tWIPE    => efectos <= wipe;
          when tPIP     => efectos <= pip;
          when tZUM     => efectos <= zum;
          when tROTATE  => efectos <= rotate;
          when tFLIP    => efectos <= flip;
          when tMOSAICO => efectos <= mosaico;
          when tSTILL   => efectos <= still1f
          when others   => efectos <= normal;
        end case;

      when pip =>
        if peticion=tCANCEL then efectos <= normal;
        else efectos <= pip;
        end if;

      when recall =>
        if peticion=tCANCEL then efectos <= normal;
        else efectos <= recall;
        end if;

      when still1f =>
        efectos <= stillrf;

      when stillrf =>
        if peticion=tCANCEL then efectos <= normal;
        else efectos <= stillrf;
        end if;

      when strobe1f =>
        efectos <= strobeRf;

      when strobeRf =>
        if (frameCounter='1') then efectos <= strobe1f;
        elsif (peticion=tCANCEL) then efectos <= normal;
        else efectos <= strobeRf;
        end if;

      when strobeC1f =>
        efectos <= strobeCRf;

      when strobeCRf =>
        if (frameCounter='1') then efectos <= strobeC1f;
        elsif (peticion=tCANCEL) then efectos <= normal;
        else efectos <= strobeCRf;
        end if;

      when wipe =>
        if peticion=tCANCEL then efectos <= normal;
        else efectos <= wipe;
        end if;

      when zum =>
        if peticion=tCANCEL then efectos <= normal;
        else efectos <= zum;
        end if;

      when rotate =>
        if peticion=tCANCEL then efectos <= normal;
        else efectos <= rotate;
        end if;

      when flip =>
        if peticion=tCANCEL then efectos <= normal;
        else efectos <= flip;
        end if;
```

```
        when others =>
          efectos <= normal;

      end case;
    end if;
end process;
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
-- Iluminamos los LEDS en funcion del efecto en el que estamos
-------------------------------------------------------------------------------
estadofsms(4) <= '1';
estadofsms(3 downto 0) <=      "0000" when efectos = mosaico
                        else "0001" when efectos = normal
                        else "0010" when efectos = pip
                        else "0011" when efectos = recall
                        else "0100" when efectos = still1f
                        else "0101" when efectos = stillrf
                        else "0110" when efectos = strobe1f
                        else "0111" when efectos = strobeRf
                        else "1000" when efectos = wipe
                        else "1001" when efectos = zum
                        else "1001" when efectos = strobec1f
                        else "1010" when efectos = strobecRf
                        else "1011" when efectos = rotate
                        else "1100" when efectos = flip
                        else "1101";
-------------------------------------------------------------------------------

end fsms_Arquitectura;
```

generadorSenyales50MHz.vhd

```
--------------------------------------------------------------------------------
-- GENERADOR DE SENYALES DE SINCRONIZACION DE VGA CON RESOLUCION 628 x
480
--   Y CONTROLADOR A 50MHz
--------------------------------------------------------------------------------
-- ENTIDAD: generadorSenyales50MHz
-- CONEXIONES:
-- PUERTOS ENTRADA:
--    relojGenerador: Entrada del reloj a 50MHz (funciona con flnaco de subida)
--    resetGenerador: Reset asincrono del circuito (activo a baja)
-- PUERTOS SALIDA:
--    hsync: Senyal de sincronizacion horizontal (de linea) (activa a baja)
--    vsync: Senyal de sincronizacion vertical (de frame) (activa a baja)
--    blanking: Senyal de blanking (activa a baja) (1:zona visible; 0:fuera)
--    pixelCont[10..0]: Contador de pixeles
--    lineaCont[9..0]: Contador de lineas
--------------------------------------------------------------------------------
-- Genera todas las senyales necesarias para el controlador, que funciona a 50MHz
-- Tiene como entradas un reloj de 50MHz y un reset asíncrono y nos da como salidas
--  las senyales de sincronizacion (horizontal, vertical, blanking) y dos contadores
--  (pixeles y lineas).
-- El contador de pixeles se incrementa en cada flanco de reloj y el de lineas
--  cada vez que hay una senyal de sincronismo de linea.
-- Las senyales de sincronizacion se activan (se ponen a cero) cuando los valores
--  de los contadores superan unas constantes predefinidas
--
-- El esquema de las senyales de sincronizacion es como aparece en el siguiente grafico
--      |<---- Region Activa ---->|<---------- Region de Blanking --------->|
--      |                        |                                         |
--   ----+--------- ... ----------+---------------          --------------+---
--   |  |                        |           |           |               |
--   |  |                        |<--Porche    |<-- Tiempo   |<-- Porche   |
--   |  |                        |  Delantero-->|    Sinc--->|  Trasero -->|
-- ---  |                        |           ----------          |
--      |                        |                                         |
--      |<---------------------- Periodo ----------------------------------->|
--
-- Para variar la configuracion de las senyales, unicamente hay que modificar
--  los valores de las constantes que corresponda.
--
-- Nota: Para conseguir la resolucion horizontal de 628, tendremos que muestrear
--  "saltandonos" el ultimo bit del contador de pixeles para que la velocidad de
--  cambio se reduzca a la mitad.
--------------------------------------------------------------------------------

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

--------------------------------------------------------------------------------
-- Declaracion de la entidad
--------------------------------------------------------------------------------
entity generadorSenyales50MHz is
  port (
    relojGenerador: in std_logic;  -- Reloj de entrada (funciona con flancos de suubida)
    resetGenerador: in std_logic;  -- Reset asincrono (activo a baja)
    hSync: out std_logic;          -- Sincronizacion horizontal (de linea) (activa a baja)
    vSync: out std_logic;          -- Sincronizacion vertical (de frame) (activa a baja)
    blanking: out std_logic;       -- Senyal de blankin (activa a baja) (1:zona visible;0:fuera)
    pixelCont: out std_logic_vector(10 downto 0); -- Contador de pixeles
    lineaCont: out std_logic_vector(9 downto 0)   -- Contador de lineas
  );
end generadorSenyales50MHz;
--------------------------------------------------------------------------------



--------------------------------------------------------------------------------
-- Declaracion de la arquitectura
--------------------------------------------------------------------------------
architecture generadorSenyales50MHz_Arquitectura of generadorSenyales50MHz is

-- Constantes para la temporizacion de las senyales:
  -- Temporizacion de linea (pixeles):
  constant H_PIXELES: INTEGER:= 1256;   -- 1256(=628x2) porque funcionamos a 50MHz)
  constant H_PORCHEDEL:   INTEGER:= 48; -- Porche delantero
  constant H_SINC: INTEGER:= 189;       -- Tiempo de sincronizacion
  constant H_PORCHETRAS: INTEGER:= 95;  -- Porche trasero
  constant H_PERIODO: INTEGER:= H_SINC + H_PIXELES + H_PORCHEDEL +
H_PORCHETRAS; -- Periodo de linea

  -- Temporizacion de frame (lineas):
  constant V_LINEAS: INTEGER:= 480;     -- 480 lineas (independientemente del numero de
pixeles)
  constant V_PORCHEDEL: INTEGER:= 14;   -- Porche delantero
```

```
   constant V_SINC: INTEGER:= 2;         -- Tiempo de sincronizacion
   constant V_PORCHETRAS: INTEGER:= 32;  -- Porche trasero
   constant V_PERIODO: INTEGER:= V_SINC + V_LINEAS + V_PORCHEDEL +
V_PORCHETRAS; --Periodo de frame

-- Contadores
   signal pixelContInt: std_logic_vector(10 downto 0);   --Contador de pixeles (horizontal)
   signal lineaContInt: std_logic_vector(9 downto 0);    --Contador de lineas (vertical)

-- Senyales auxiliares (hacen falta porque no podemos leer de los puertos de salida)
   signal hSyncAux: std_logic;
   signal vSyncAux: std_logic;
   signal blankingAux: std_logic;

--Relojes axiliares:
   signal reloj25MHz: std_logic;   -- Reloj auxiliar a 25Mhz
   signal reloj125MHz: std_logic;  -- Reloj auxiliar a 12'5Mhz


begin


-----------------------------------------
-- Contador de pixeles
-----------------------------------------
-- Es un contador modulo H_PERIODO que se incrementa con cada flanco de
--  subida del reloj de entrada.
-----------------------------------------
ContadorPixeles:
process(relojGenerador, resetGenerador)
begin
   -- El resetGenerador a cero inicializa el valor del contador
   if (resetGenerador = '0') then
      pixelContInt <= (others => '0');
   elsif (relojGenerador'event and relojGenerador = '1') then
      -- Solo responde ante flanco de subida del reloj
      -- Se incrementara hasta llegar al valor del periodo horizontal (num. pixeles)
      if (pixelContInt < H_PERIODO) then
         pixelContInt <= pixelContInt + 1;
      else
         pixelContInt <= (others => '0');
      end if;
   end if;
end process;
-----------------------------------------
-----------------------------------------
-- Contador de lineas
-----------------------------------------
-- Es un contador modulo H_LINEAS que se incrementa con cada flanco de
--  subida de la senyal de sincronizacion horizontal (cada vez que hemos
--  terminado de mostrar una linea).
-----------------------------------------
ContadorLineas:
process(hSyncAux, resetGenerador)
begin
   -- El resetGenerador a cero inicializa el valor del contador
   if (resetGenerador='0') then
      lineaContInt <= (others => '0');
   elsif (hSyncAux'event and hSyncAux = '1') then
      -- Solo responde ante flanco de subida de la senyal de sincronizacion horizontal
      -- Se incrementara hasta llegar al valor del periodo vertical (num. lineas)
      if (lineaContInt < V_PERIODO) then
         lineaContInt <= lineaContInt + 1;
      else
         lineaContInt <= (others => '0');
      end if;
   end if;
end process;
-----------------------------------------


-----------------------------------------
-- Implementacion de la senyal de sincronizacion horizontal (de linea)
-----------------------------------------
-- Depende del valor del contador de pixels y se activara cuando el contador
--  haya superado el valor del porche delantero mas el numero de pixeles visibles.
--------------------
SincronizaciónHorizontal:
process(relojGenerador, resetGenerador)
begin
   -- El resetGenerador a cero la desactiva (la pone a uno)
   if (resetGenerador = '0') then
      hSyncAux <= '1';
   -- Responde ante los flancos de subida del reloj
   elsif (relojGenerador'event and relojGenerador = '1') then
      -- Dependiendo del intervalo, se activa para sincronizar o no e indicar una nueva linea
      if (pixelContInt >= (H_PORCHEDEL + H_PIXELES) and pixelContInt < (H_PIXELES +
H_PORCHEDEL + H_SINC)) then
         hSyncAux <= '0';
      else
         hSyncAux <= '1';
```

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


----------------------------------------
-- Implementacion de la senyal de sincronizacion vertical (de frame)
----------------------------------------
-- Depende del valor del contador de linea y se activara cuando el contador
--  haya superado el valor del porche delantero mas el numero de lineas visibles.
----------------------------------------
SincronizacionVertical:
process(hSyncAux, resetGenerador)
begin
  -- El resetGenerador a cero la desactiva (la pone a uno)
  if (resetGenerador = '0') then
    vSyncAux <= '1';
  -- Responde ante los flancos de subida de la senyal de sincronizacion horizontal
  elsif (hsyncAux'event and hSyncAux = '1') then
    -- Dependiendo del intervalo, se activa para sincronizar o no e indicar un nuevo frame
    if (lineaContInt >= (V_LINEAS + V_PORCHEDEL) and lineaContInt < (V_LINEAS +
V_PORCHEDEL + V_SINC)) then
      vSyncAux <= '0';
    else
      vSyncAux <= '1';
    end if;
  end if;
end process;
----------------------------------------


----------------------------------------
-- Implementacion de la senyal de blanking
----------------------------------------
-- Depende del valor del contador de pixels y del de lineas.
-- Activa a cero. Cuando estemos fuera de la zona visible de la pantalla, hay
--  que hacer blanking, por lo que la ponemos a cero. En zona visible, vale uno.
----------------------------------------
senyalDeBlanking:
process (relojGenerador, resetGenerador)
begin
  if (resetGenerador = '0') then
    blankingAux <= '0';
  elsif (relojGenerador'EVENT and relojGenerador = '1') then
    -- Cero cuando estamos fuera y uno cuando estamos dentro
    if (pixelContInt >= H_PIXELES or lineaContInt >= V_LINEAS) then
      blankingAux <= '0';
    else
      blankingAux <= '1';
    end if;
  end if;
end process;
----------------------------------------


----------------------------------------
-- Sacamos todas las senyales que hemos estado generando
----------------------------------------
  hSync <= hSyncAux;          -- Sincronización horizontal
  vSync <= vsyncAux;          -- Sincronización vertical
  blanking <= blankingAux;    -- Blanking
  pixelCont <= pixelContInt;  -- Contador de pixeles
  lineaCont <= lineaContInt;  -- Contador de lineas
----------------------------------------


end generadorSenyales50MHz_Arquitectura;
```

interfazRam.vhd

```
--------------------------------------------------------------------------------
-- INTERFAZ DE ACCESO A UNA MEMORIA RAM DE BUFFER UNICO
--------------------------------------------------------------------------------
-- ENTIDAD: interfazRam
-- CONEXIONES:
-- PUERTOS ENTRADA:
--    relojInterfaz: std_logic -- Reloj de entrada
--    resetInterfaz: std_logic -- Reset del circuito (activo a baja)
--    hazLectura: std_logic    -- Senyal para solicitar una escritura (activa a alta)
--    hazEscritura: std_logic  -- Senyal para solicitar una escritura (activa a alta)
--    direccionDeLectura[18..0]: std_logic_vector   --Direccion de la que quremos leer
--    direccionDeEscritura[18..0]: std_logic_vector -- Direccion sobre la que queremos
escribir
--    datoAEscribir[15..0]: std_logic_vector       -- Dato que queremos escribir en la memoria
(solo entrada)
-- PUERTOS SALIDA:
--    datoLeido[15..0]: std_logic_vector       -- Dato que hemos leido de la memoria (solo
salida)
--    chipEnable: std_logic                    -- Chip Enable de la memoria (activo a baja)
--    outputEnable: std_logic                   -- Output Enable de la memoria (activo a baja)
(prioritario)
--    writeEnable: std_logic                   -- Write Enable de la memoria (activo a baja)
--    direccionMemoria[18..0]: std_logic_vector -- Bus de direcciones que va a la memoria
-- PUERTOS ENTRADA/SALIDA:
--    datosMemoria[15..0]: std_logic_vector     -- Bus de datos bidireccional que se
comunica con la memoria
--------------------------------------------------------------------------------
-- Es un interfaz que nos permite acceder a una memoria Ram de bus unico.
-- Implementa los accesos a esta en un unico ciclo (tanto lectura como escritura) por lo
que se pueden alternar
--  los accesos en ciclos consecutivos si asi se requiere.
-- Ademas y, principalmente para mejorar la temporizacion en el acceso, se ha
implementado una estrutura de doble
--  buffer, tanto para las direcciones como para los datos, de modo que disponemos de
manera independiente de una
--  conexion en sentido salida (lectura) y otra en sentido entrada (escritura), que ya se
multiplexaran dentro
--  del modulo de manera adecuada pero que permiten a los dispositivos que se conectan
a el el disponer de mas tiempo
-- Tanto los datos como las direcciones se registran internamente, por lo que no es
necesario que lo hagan los modulos
--  externos que se conecten con el interfaz. Sin embargo, es muy conveniente que los que
lo vayan a utilzar para
--  lectura, registren el puerto de datos leídos para asegurar que cogen el dato que
solicitaron.
-- El proceso, tanto para lectura como escritura consiste en realizar una peticion de
operacion poniendo la senyal
--  correspondiente a alta y los datos y direccion con los que queremos que se opere.
Como la operacion se hara efectiva
--  en el siguiente ciclo y los registros no se cargaran hasta el flanco de reloj, el orden en
que fijemos las senyales
--  es totalmente independiente, con lo que dotamos al interfaz de gran flexibilidad.
--------------------------------------------------------------------------------

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

--------------------------------------------------------------------------------
-- Inicio de la entidad
--------------------------------------------------------------------------------
entity interfazRam is
  port (
    relojInterfaz: in std_logic; -- Reloj de entrada
    resetInterfaz: in std_logic; -- Reset del circuito (activo a baja)
    hazLectura: in std_logic;    -- Senyal para solicitar una escritura (activa a alta)
    hazEscritura: in std_logic;  -- Senyal para solicitar una escritura (activa a alta)
    direccionDeLectura: in std_logic_vector (18 downto 0);   --Direccion de la que quremos leer
    direccionDeEscritura: in std_logic_vector (18 downto 0); -- Direccion sobre la que queremos
escribir
    datoLeido: out std_logic_vector (15 downto 0);    -- Dato que hemos leido de la memoria (solo
salida)
    datoAEscribir: in std_logic_vector (15 downto 0); -- Dato que queremos escribir en la memoria
(solo entrada)
    chipEnable: out std_logic;   -- Chip Enable de la memoria (activo a baja)
    outputEnable: out std_logic; -- Output Enable de la memoria (activo a baja) (prioritario)
    writeEnable: out std_logic;  -- Write Enable de la memoria (activo a baja)
    direccionMemoria: out std_logic_vector (18 downto 0);   -- Bus de direcciones que va a la
memoria
    datosMemoria: inout std_logic_vector (15 downto 0)      -- Bus de datos bidireccional que se
comunica con la memoria
  );
end interfazRam;
```

```
-------------------------------------------------------------------------------
-- Inicio de la arquitectura
-------------------------------------------------------------------------------
architecture interfazRam_arquitectura of interfazRam is

  -- Senyales para los registros:
  signal regDireccion : std_logic_vector(18 downto 0);     -- Direccion que usaremos para acceder
a la memoria (lectura o escritura)
  signal regDatoAEscribir : std_logic_vector(15 downto 0); -- Registro que hemos leido y/o vamos
a escribir en la memoria

  -- Senyales de carga para los registros (activas a alta):
  signal cRegdireccionDeEscritura : std_logic;
  signal cRegdireccionDeLectura : std_logic;
  signal cRegDatoAEscribir : std_logic;

  -- Valores internos para las senyales de control de la memoria (activas a baja):
  signal CEn : std_logic;
  signal OEn : std_logic;
  signal WEn : std_logic;

  -- Para saber que estamos haciendo en ese ciclo (activas a alta)
  signal haciendoEscritura : std_logic;
  signal guardaDatoLeido : std_logic;

  -- Bus de 16 bits para las comunicaciones:
  signal busDatosRam : std_logic_vector(15 downto 0);

  -- Senyales para las salidas Mealy:
  signal siguienteOEn: std_logic;
  signal siguienteWEn: std_logic;
  signal siguienteHaciendoEscritura: std_logic;
  signal siguienteGuardaDatoLeido : std_logic;

  -- Senyales para la maquina de estados:
  type TEstado is (stEnEspera, stEscribe, stLee);
  signal estadoActual: TEstado;
  signal estadoSiguiente: TEstado;

begin

  --Asignaciones concurrentes
    -- Senyales de control de la memoria
    chipEnable <= CEn;
    outputEnable <= OEn;
    writeEnable <= WEn;
    -- Buses de la memoria (salen desde registros)
    direccionMemoria <= regDireccion;
    datosMemoria <= busDatosRam;


-------------------------------------------------------------------------------
-- Inicio de la implementacion del resto de estructuras
-------------------------------------------------------------------------------

  -- Buffer triestado colocado despues del registro regDatoAEscribir en el bus de datos
  -- Si estamos escribiendo volcamos el contenido del registro y si estamos leyendo,
  --  lo ponemos en alta impedancia para poder leer
  elBufferDeDatos:
  process(regDatoAEscribir, haciendoEscritura, relojInterfaz)
  begin
    if (haciendoEscritura = '1') then
      busDatosRam <= regDatoAEscribir;
    else
      busDatosRam <= (others => 'Z');
    end if;
  end process;


  -- Proceso para la señal de escritura. Se independiza del reloj para poder adaptar su
temporizacion y asegurar que
  --  baja despues de que direccion y datos estn estables y la escritura sea correcta
  laSenyalDeEscritura:
  process(haciendoEscritura, relojInterfaz)
  begin
    if (haciendoEscritura = '1') then
      WEn <= '0';
    else
      WEn <= '1';
    end if;
  end process;


  -- Proceso para el registro de los datos leidos de la memoria. En el caso de que la senyal de
carga se haya activado
  --  tenemos que almacenar el valor del bus porque va a haber una escritura y se machacarian
los datos que queremos leer.
  elGuardaDatoLeido:
  process(guardaDatoLeido, resetInterfaz)
  begin
    if (resetInterfaz='0') then
      datoLeido <= (others => '0');
```

```
  elsif (guardaDatoLeido'event and guardaDatoLeido='1') then
    datoLeido <= busDatosRam;
  end if;
end process;


-- Proceso para los eventos dependientes del flanco de reloj
elProcesoClockeado:
process(relojInterfaz, resetInterfaz)
begin
  if (resetInterfaz = '0') then
    -- Valores por defecto para las senyales:

 estadoActual <= stEnEspera;

 CEn <= '1';
    OEn <= '0';

 haciendoEscritura <= '0';
    guardaDatoLeido <= '0';

 regDireccion <= (others => '0');
    regDatoAEscribir <= (others => '0');

  elsif (relojInterfaz'EVENT and relojInterfaz = '1') then

    CEn <= '0';  -- Siempre está capacitada

    -- Gestionamos las senyales de carga de los registros de los datos:
    if (cRegdireccionDeEscritura = '1') then
      regDireccion <= direccionDeEscritura;
    end if;
    if (cRegdireccionDeLectura = '1') then
      regDireccion <= direccionDeLectura;
    end if;

    -- Para mantener estables los datos a escribir en memoria, usamos un registro:
    if (cRegDatoAEscribir = '1') then
      regDatoAEscribir <= datoAEscribir;
    end if;

    -- Actualizamos las senyales Mealy
    OEn <= siguienteOEn;
    haciendoEscritura <= siguientehaciendoEscritura;
    guardaDatoLeido <= siguienteGuardaDatoLeido;

    -- Actualizamos el estado de la maquina
    estadoActual <= estadoSiguiente;

  end if;
end process;


-- Proceso para el controlador de la Maquina de Estados
elControladorFSM:
process(estadoActual, hazLectura, hazEscritura)
begin
  -- Asignacion de los valores de las senyales por defecto
  -- Senyales Mealy
  siguienteOEn <= '0';
  siguientehaciendoEscritura <= '0';
  siguienteguardaDatoLeido <= '0';

  -- Senyales de carga de registros:
  cRegdireccionDeEscritura <= '0';
  cRegDatoAEscribir <= '0';
  cRegdireccionDeLectura <= '0';

  case estadoActual is

    when stEnEspera =>
      -- Estamos esperando a que nos hagan una peticion:
      estadoSiguiente <= stEnEspera;

      if (hazEscritura = '1') then
        -- Peticion de escritura
        estadoSiguiente <= stEscribe;

        cRegdireccionDeEscritura <= '1';
        cRegDatoAEscribir <= '1';

        siguienteOEn <= '1';
        siguienteHaciendoEscritura <= '1';
        siguienteGuardaDatoLeido <= '1';
      elsif (hazLectura = '1') then
        -- Peticion de lectura
        estadoSiguiente <= stLee;

        cRegdireccionDeLectura <= '1';
      else
        --No hay peticion
        siguienteGuardaDatoLeido <= '1';
```

```
    end if;

  when stEscribe =>
    estadoSiguiente <= stEnEspera;

    siguienteOEn <= '0';
    siguientehaciendoEscritura <= '0';
    siguienteguardaDatoLeido <= '0';

    if (hazEscritura = '1') then
      -- Peticion de escritura
      estadoSiguiente <= stEscribe;

      cRegdireccionDeEscritura <= '1';
      cRegDatoAEscribir <= '1';

      siguienteOEn <= '1';
      siguienteHaciendoEscritura <= '1';
      siguienteGuardaDatoLeido <= '1';
    elsif (hazLectura = '1') then
      -- Peticion de lectura
      estadoSiguiente <= stLee;

      cRegdireccionDeLectura <= '1';
    else
      --No hay peticion
      siguienteGuardaDatoLeido <= '1';

    end if;

  when stLee =>
    estadoSiguiente <= stEnEspera;

    if (hazEscritura = '1') then
      -- Peticion de escritura
      estadoSiguiente <= stEscribe;

      cRegdireccionDeEscritura <= '1';
      cRegDatoAEscribir <= '1';

      siguienteOEn <= '1';
      siguienteHaciendoEscritura <= '1';
      siguienteGuardaDatoLeido <= '1';
    elsif hazLectura = '1' then
      -- Peticion de lectura
      estadoSiguiente <= stLee;

      cRegdireccionDeLectura <= '1';
    else
      --No hay peticion
      siguienteGuardaDatoLeido <= '1';
    end if;

  end case;

end process;

end interfazRam_arquitectura;
```

ajustes.vhd

```
--------------------------------------------------------------------------------
-- MODULO QUE IMPLEMENTA LOS COMPONENTES IMPICADOS EN LOS AJUSTES DE
LOS EFECTOS
--------------------------------------------------------------------------------
-- ENTIDAD: Ajustes
-- CONEXIONES:
-- PUERTOS ENTRADA:
--   relojAjustes: Reloj general del sistema (50 MHz)
--   resetAjustes: Reset general del sistema (activo a baja)
--   vSync: Senyal de sincronización horizontal
--   tAjuste(1..0): Codificación del tipoe de ajuste
--   tPeticion(3..0): Codificación del tipo de efecto
--   dirRefrescoBase(18..0): Direccion que se usaria en el refresco normal
--   dirRefrescoBase(18..0): Direccion que se uso en el ciclo anterior
-- PUERTOS SALIDA:
--   esZonaWipe: Indica si estamos en la zona estatica o no (activa a alta)
--   esZonaPip: Indica si estamos dentro del cuadrado o no (activa a alta)
--   frameCounter: Fin de cuenta del contador de frames de los estrobes (activo a alta)
--   dirPip(18..0): Dirección precalculada para el efecto pip
--   dirZoom(18..0): Dirección precalculada para el efecto zoom
--   dirMosaico(18..0): Dirección precalculada para el efecto mosaico
--   dirStrobe(18..0): Dirección precalculada para el efecto estrobe
--   dirFlip(18..0): Dirección precalculada para el efecto reflejo
--   dirRotacion(18..0): Dirección precalculada para el efecto rotacion
--------------------------------------------------------------------------------
-- Este modulo implementa todos los componentes que puedan hacer falta para
--  gestionar las posibles modificaciones que se puedanb hacer sobre los efectos
--  ademas de encrgarse de componer las direcciones que se usan para acceder a
--  las memorias y de generar determinadas senyales que permitiran al controlador
--  saber como tiene que actuar.
-- Como entradas tiene el tipo de efecto que estamos visualizando, el tipo de
--  ajuste a realizar (en caso de que lo haya) y las dos direcciones base.
-- A partir de esto, nos entrega todas las direcciones que nos puedan hacer falta
--  y que se seleccionara en el controlador.
-- Se ha agrupado todo en funcion del efecto al que corresponde por claridad
--------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

library SYNOPSYS;
use SYNOPSYS.attributes.all;


--------------------------------------------------------------------------------
-- Declaracion de la entidad
--------------------------------------------------------------------------------
entity Ajustes is
  port (
    relojAjustes: in std_logic;
    resetAjustes: in std_logic;
    vSync: in std_logic;
    tAjuste : in std_logic_vector (1 downto 0);
    tPeticion : in std_logic_vector (3 downto 0);
    dirRefrescoBase: in std_logic_vector (18 downto 0);
    dirRefrescoReg : in std_logic_vector (18 downto 0);
    esZonaWipe: out std_logic;
    esZonaPip: out std_logic;
    frameCounter: out std_logic;
    dirPip: out std_logic_vector( 18 downto 0 );
    dirZoom: out std_logic_vector( 18 downto 0 );
    dirMosaico: out std_logic_vector( 18 downto 0 );
    dirStrobeC: out std_logic_vector( 18 downto 0 );
    dirFlip: out std_logic_vector( 18 downto 0 );
    dirRotacion: out std_logic_vector( 18 downto 0 )
  );
end Ajustes;
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
-- Declaracion de la arquitectura
--------------------------------------------------------------------------------
architecture Ajustes_Arquitectura of Ajustes is
```

```
-------------------------------------------------------------------------------
-- Senyales para las rotaciones y el reflejo
-------------------------------------------------------------------------------
  type modo_Rotacion is ( nr, r90, r180, r270 );
  signal modoRotacion: modo_Rotacion;

  type modo_Flip is ( nf, fv, fh, fvh);
  signal modoFlip: modo_Flip;

  signal esZonaRotacion: std_logic;
  signal resta480_Y: std_logic_vector(9 downto 0);
  signal resta480_X: std_logic_vector(9 downto 0);
  signal resta628_Y: std_logic_vector(9 downto 0);
  signal resta628_X: std_logic_vector(9 downto 0);
--  signal resta480_Y: std_logic_vector(8 downto 0);
--  signal resta480_X: std_logic_vector(8 downto 0);
--  signal resta628_Y: std_logic_vector(9 downto 0);
--  signal resta628_X: std_logic_vector(9 downto 0);
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
-- Senyales para el estrobe ciclico
-------------------------------------------------------------------------------
  -- Tiempo en numero de frames que dura el strobe
  signal tiempoStrobe: std_logic_vector (8 downto 0);
  -- Contador de frames
  signal sFrameCounter: std_logic_vector (8 downto 0);
  signal finFrameCounter: std_logic;
  -- En que cuadrante estamos
  signal cuadranteStrobeC: std_logic_vector (1 downto 0);
  -- Para ver si estamos dentro de la zona a almacenar o no
  signal esZonaXStrobeC: std_logic;
  signal esZonaYStrobeC: std_logic;
  signal esZonaStrobeC: std_logic;
  -- Coordenadas del origen del cuadrante al que corresponde
  signal strobeC_OrigenX: std_logic_vector (9 downto 0);
  signal strobeC_OrigenY: std_logic_vector (8 downto 0);
  -- Componentes para operar con la direccion por separado
  signal strobeC_DireccionX: std_logic_vector (9 downto 0);
  signal strobeC_DireccionY: std_logic_vector (8 downto 0);
  -- La direccion en si
  signal direccionStrobeCiclico: std_logic_vector (18 downto 0);
-------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------
-- Senyales para el pip
-------------------------------------------------------------------------------
  -- Registro para ver en que cuadrante esta el pip
  signal cuadrantePip: std_logic_vector (1 downto 0); -- El valor del cuadrante en gray
  -- Para ver si estamos dentro de la zona  o no
  signal esZonaXPip: std_logic;
  signal esZonaYPip: std_logic;
  signal dentroZonaPip: std_logic;
  -- Coordenadas del origen del cuadrante al que corresponde
  signal pip_OrigenX: std_logic_vector (9 downto 0);
  signal pip_OrigenY: std_logic_vector (8 downto 0);
  -- Componentes para operar con la direccion por separado
  signal pip_DireccionX: std_logic_vector (9 downto 0);
  signal pip_DireccionY: std_logic_vector (8 downto 0);
  -- La direccion en si
  signal direccionPip: std_logic_vector (18 downto 0);
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
-- Senyales para el wipe
  -- Se  usara para las comparaciones, para determinar que parte de la pantalla corresponde a
cada banco
  signal contadorWipe :std_logic_vector(9 downto 0);
  signal cargaContadorWipe :std_logic_vector (1 downto 0);
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
-- Senyales para el zoom
-------------------------------------------------------------------------------
  signal zoomx2: std_logic_vector(18 downto 0);      -- Direccion del zoom  x2
  signal zoomx4: std_logic_vector(18 downto 0);      -- Direccion del zoom  x4
  signal zoomd2: std_logic_vector(18 downto 0);      -- Direccion del zoom  1/2
  signal muxZoom : std_logic_vector (18 downto 0);  -- Direccion que va a la maquina
  signal contadorZoom: std_logic_vector(1 downto 0);-- Salida del contador modulo 4 que
selecciona el tamano del zoom
  signal cuentaZoom: std_logic;           -- Senyal de cuenta del contador zoom
  signal esZonaZoom: std_logic;
-------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------
-- Senyales del mosaico
-------------------------------------------------------------------------------
   signal dirMosaicoLinea: std_logic_vector(8 downto 0);   -- Parte de la dir del mosaico
correspondiente a las lineas
   signal dirMosaicoPixel: std_logic_vector(9 downto 0); -- Parte de la dir del mosaico
correspondiente a los pixeles
   signal regMascaraLinea: std_logic_vector(8 downto 0); -- Mascara de las lineas
   signal regMascaraPixel: std_logic_vector(9 downto 0); -- Mascara de los pixeles
   signal dirMosaicoAux: std_logic_vector(18 downto 0);  -- Direccion del mosaico
   signal modificaMascara: std_logic;         -- Senyal para poder modificar la mascara
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
-- Declaracion de constantes simbolicas
-------------------------------------------------------------------------------
   -- Para los efectos
   constant tCANCEL  : std_logic_vector (3 downto 0) := "0000";
   constant tSTILL   : std_logic_vector (3 downto 0) := "0001";
   constant tRECALL  : std_logic_vector (3 downto 0) := "0010";
   constant tSTROBE  : std_logic_vector (3 downto 0) := "0011";
   constant tWIPE    : std_logic_vector (3 downto 0) := "0100";
   constant tPIP     : std_logic_vector (3 downto 0) := "0101";
   constant tZUM     : std_logic_vector (3 downto 0) := "0110";
   constant tMOSAICO : std_logic_vector (3 downto 0) := "0111";
   constant tFLIP    : std_logic_vector (3 downto 0) := "1000";
   constant tROTATE  : std_logic_vector (3 downto 0) := "1001";
   constant tSTROBEC : std_logic_vector (3 downto 0) := "1010";

   -- Para los ajustes
   constant tNone    : std_logic_vector (1 downto 0) := "00";
   constant tMas     : std_logic_vector (1 downto 0) := "01";
   constant tMenos   : std_logic_vector (1 downto 0) := "11";

   -- Para las dimensiones de la pantalla
   constant KAncho_628  : std_logic_vector (9 downto 0) := "1001110100";  -- 628
   constant KAlto_480   : std_logic_vector (8 downto 0) := "111100000";   -- 480
   constant KMitadX_314 : std_logic_vector (9 downto 0) := "0100111010" ; -- 314
   constant KMitadY_240 : std_logic_vector (8 downto 0) := "011110000";   -- 240
-------------------------------------------------------------------------------


begin
```

```
-------------------------------------------------------------------------------
-- Restas que pueden hacer falta para la rotacion y el flip
-- Las hacemos siempre y luego las usaremos o no
-------------------------------------------------------------------------------
   resta480_Y <= KAlto_480 - ("0" & dirRefrescoBase(18 downto 10));
   resta480_X <= KAlto_480 - (dirRefrescoBase(9 downto 0));
   resta628_Y <= KAncho_628 - ("0" & dirRefrescoBase(18 downto 10));
   resta628_X <= KAncho_628 - (dirRefrescoBase(9 downto 0));
--      resta480_Y <= KAlto_480 - dirRefrescoBase(18 downto 10);
--      resta480_X <= KAlto_480 - dirRefrescoBase(8 downto 0);
--      resta628_Y <= KAncho_628 - ("0" & dirRefrescoBase(18 downto 10));
--      resta628_X <= KAncho_628 - (dirRefrescoBase(9 downto 0));
-------------------------------------------------------------------------------


-------------------------------------------------------------
-----  Inicio de la Rotación
-------------------------------------------------------------
-- Proceso para comprobar que tipo de rotacion tenemos que aplicar
-- El orden es 0, 90, 180, 270 grados
SiguienteCicloRotacion:
process (vSync, resetAjustes, tAjuste, tPeticion)
begin
   if resetAjustes='0' then
      modoRotacion <= nr;
   elsif (vSync'event and vSync= '1') then
      modoRotacion <= modoRotacion;
      if (tPeticion = tRotate) then
         case modoRotacion is
            when nr => -- No hay rotacion (0 grados)
               if tAjuste = tMas then
                  modoRotacion <= r90;
               elsif tAjuste = tMenos then
                  modoRotacion <= r270;
               else
                  modoRotacion <= nr;
               end if;
            when r90 => -- 90 grados
               if tAjuste = tMas then
                  modoRotacion <= r180;
               elsif tAjuste = tMenos then
                  modoRotacion <= nr;
               else
                  modoRotacion <= r90;
               end if;
            when r180 => -- 180 grados
```

```
            if tAjuste = tMas then
              modoRotacion <= r270;
            elsif tAjuste = tMenos then
              modoRotacion <= r90;
            else
              modoRotacion <= r180;
            end if;
          when r270 => -- 270 grados
            if tAjuste = tMas then
              modoRotacion <= nr;
            elsif tAjuste = tMenos then
              modoRotacion <= r180;
            else
              modoRotacion <= r270;
            end if;
        end case;
      end if;
  end if;
end process;

-- Proceso para calcular la direccion que se usara en funcion del modo
salidasRotacion:
process (modoRotacion, dirRefrescoBase, resta480_Y, resta480_X, resta628_Y, resta628_X)
begin
  case modoRotacion is
    when nr =>
      dirRotacion <= dirRefrescoBase(18 downto 10) & dirRefrescoBase(9 downto 0);
    when r90 =>
      if (dirRefrescoBase(9 downto 0) < KAlto_480) then
        dirRotacion <= resta480_X(8 downto 0) & "0" & dirRefrescoBase(18 downto 10);
--        dirRotacion <= resta480_X & "0" & dirRefrescoBase(18 downto 10);
      else
        dirRotacion <= (others => '0');
      end if;
    when r180 =>
      dirRotacion <= resta480_Y(8 downto 0) & resta628_X;
--      dirRotacion <= resta480_Y & resta628_X;
    when r270 =>
      if (dirRefrescoBase(9 downto 0) < KAlto_480) then
        dirRotacion <= dirRefrescoBase(8 downto 0) & resta628_Y;
--        dirRotacion <= dirRefrescoBase(8 downto 0) & resta628_Y;
      else
        dirRotacion <= (others => '0');
      end if;
  end case;
end process;
```

```
-------------------------------------------------------------
-----  Inicio del Reflejo
-------------------------------------------------------------
-- Proceso para ver a que modo pasamos en funcion de la tecla pulsada
-- El orden es normal, vertical, vertical-horizontal, horizontal
SiguienteCicloFlip:
process (vSync, resetAjustes, tAjuste, tPeticion)
begin
  if (resetAjustes = '0') then
    modoFlip <= nf;
  elsif (vSync'event and vSync= '1') then
    modoFlip <= modoFlip;
    if (tPeticion = tFlip) then
      case modoFlip is
        when nf => -- Modo normal (sin reflejo)
          if tAjuste = tMas then
            modoFlip <= fv;
          elsif tAjuste = tMenos then
            modoFlip <= fh;
          else
            modoFlip <= nf;
          end if;
        when fv => -- Reflejo vertical
          if tAjuste = tMas then
            modoFlip <= fvh;
          elsif tAjuste = tMenos then
            modoFlip <= nf;
          else
            modoFlip <= fv;
          end if;
        when fh => -- Reflejo horizontal
          if tAjuste = tMas then
            modoFlip <= nf;
          elsif tAjuste = tMenos then
            modoFlip <= fvh;
          else
            modoFlip <= fh;
          end if;
        when fvh => -- Reflejo vertical y horizontal
          if tAjuste = tMas then
            modoFlip <= fh;
          elsif tAjuste = tMenos then
            modoFlip <= fv;
          else
            modoFlip <= fvh;
          end if;
```

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

-- Direccion que usaremos en función del modo
salidasFlip:
process (modoFlip, dirRefrescoBase, resta480_Y, resta480_X, resta628_Y, resta628_X )
begin
  case modoFLip is
    when nf =>
      dirFlip <= dirRefrescoBase(18 downto 10) & dirRefrescoBase(9 downto 0);
    when fv =>
      dirFlip <= resta480_Y(8 downto 0) & dirRefrescoBase(9 downto 0);
    when fh =>
      dirFlip <= dirRefrescoBase(18 downto 10) & resta628_X;
    when fvh =>
      dirFlip <= resta480_Y( 8 downto 0) & resta628_X;
  end case;
end process;
-------------------------------------------------------------
-----  Fin del Reflejo
-------------------------------------------------------------



-------------------------------------------------------------
-----  Inicio del Estrobe Ciclico
-------------------------------------------------------------
-- Registro en el que tenemos el tiempo (en numero de frames) del estrobe
-- Para aumentar tiempo, desplazar a la izquierda metiendo unos
-- Para disminuir, desplazar a la derecha metiendo ceros
elTiempoStrobe:
process(resetAjustes, vSync, tPeticion, tAjuste)
begin
  if (resetAjustes = '0') then
    tiempoStrobe <= "000111111";  -- Valor por defecto
  elsif (vSync'event and vSync='1') then
    tiempoStrobe <= tiempoStrobe;
    -- Solo modificaremos si estamos en uno de los strobes
    if (tPeticion=tSTROBE or tPeticion=tSTROBEC) then
      case (tAjuste) is
        when tMas =>
          tiempoStrobe <= tiempoStrobe(7 downto 0) & "1";  -- <<1 (*2)
        when tMenos =>
          tiempoStrobe <= "0" & tiempoStrobe(8 downto 1);  -- >>1 (/2)
        when others =>
          tiempoStrobe <= tiempoStrobe;  --Sigue igual
      end case;
    end if;
  end if;
end process;

-- Contador descendente que cuando llega a cero se vuelve a cargar con el contenido del
temporizador
-- Tiene una senyal de fin de cuenta que se activa (a alta) cuando llega a cero
elFameCounter:
process(resetAjustes, vSync, tiempoStrobe)
begin
  -- El reset inicializa el valor del contador con el valor que haya en el registro base
  if (resetAjustes = '0') then
    sFrameCounter <= tiempoStrobe;
  elsif (vSync'event and vSync='1') then
    -- Se decrementa hasta llegar a cero con pulsos de vsync (cada frame)
    if (sFrameCounter = 0) then
      finFrameCounter <= '1';
      sFrameCounter <= tiempoStrobe;
    else
      sFrameCounter <= sFrameCounter - 1;
      finFrameCounter <= '0';
    end if;
  end if;
end process;

--Contador para ir alternando los cuadrantes
--Se ha hecho gray por conveniencia a la hora de usarlo como senyal de seleccion
elContadorDelCuadranteDelStrobeCiclico:
process (resetAjustes, FinFrameCounter)
begin
  if (resetAjustes = '0') then
    cuadranteStrobeC <= "11";
  elsif (FinFrameCounter'EVENT and FinFrameCounter = '1') then
    case (cuadranteStrobeC) is
      when "00" => cuadranteStrobeC <= "01";
      when "01" => cuadranteStrobeC <= "11";
      when "11" => cuadranteStrobeC <= "10";
      when "10" => cuadranteStrobeC <= "00";
      when others => cuadranteStrobeC <= "11";
    end case;
  end if;
end process;
```

```
-- Seeleccionamos el origen de los cuadrantes en funcion del valor del contador de los
cuadrantes
strobeC_OrigenY <= KMitadY_240 when cuadranteStrobeC(1)='1'  -- Cuadrantes 2 y 3 (abajo)
                else (others => '0');                -- Cuadrantes 0 y 1 (arriba)
strobeC_OrigenX <= KMitadX_314 when cuadranteStrobeC(0)='1'  -- Cuadrantes 1 y 3 (derecha)
                else (others => '0');                -- Cuadrantes 0 y 2 (izquierda)

-- Componemos la direccion como corresponda (zoom x1/2 y recolocacion)
-- Quitamos el bit menos significativo de cada componente metiendo un cedo por la izquierda
(zoom x1/2)
-- y le sumamos el origen del cuadrante al que corresponde
strobeC_DireccionY <= ("0" & dirRefrescoReg(18 downto 11)) + strobeC_OrigenY;
strobeC_DireccionX <= ("0" & dirRefrescoReg(9 downto 1)) + strobeC_OrigenX;

-- Vemos si estamos dentro de las coordenadas de ese cuadrante o no
esZonaYStrobeC <= '1' when dirRefrescoReg(18 downto 10)<KAlto_480
                  else '0';
esZonaXStrobeC <= '1' when dirRefrescoReg(9 downto 0)<KAncho_628
                  else '0';
esZonaStrobeC <= esZonaYStrobeC and esZonaXStrobeC;

-- La direccion del strobe ciclico es la union de ambas
-- Si estamos fuera de zona, le ponemos la cero para que escriba basura y no machaque otras
posiciones
direccionStrobeCiclico <= strobeC_DireccionY & strobeC_DireccionX when esZonaStrobeC='1'
                     else (others => '0');

-- Asignaciones concurrentes (salidas que dara el modulo)
dirStrobeC <= direccionStrobeCiclico;
frameCounter <= finFrameCounter;
-------------------------------------------------------------
-----  Fin del Estrobe Ciclico
-------------------------------------------------------------



-------------------------------------------------------------
-----  Inicio del Pip
-------------------------------------------------------------
-- Registro en el que tenemos el cuadrante sobre el que mostramos el pip
-- Es un contador gray que se incrementa decrementa en funcion de la pulsacion
elCuadratePip:
process(resetAjustes, vSync, tPeticion, tAjuste, cuadrantePip)
begin
  if (resetAjustes = '0') then
     cuadrantePip <= "11";
  elsif (vSync'EVENT and vSync = '1') then
     cuadrantePip <= cuadrantePip;
     if (tPeticion=tPIP) then
       case (tAjuste) is
         when tMas =>
           case (cuadrantePip) is
             when "00" => cuadrantePip <= "01";
             when "01" => cuadrantePip <= "11";
             when "11" => cuadrantePip <= "10";
             when "10" => cuadrantePip <= "00";
             when others => cuadrantePip <= "11";
           end case;
         when tMenos =>
           case (cuadrantePip) is
             when "00" => cuadrantePip <= "10";
             when "10" => cuadrantePip <= "11";
             when "11" => cuadrantePip <= "01";
             when "01" => cuadrantePip <= "00";
             when others => cuadrantePip <= "11";
           end case;
         when others =>
           cuadrantePip <= cuadrantePip;
       end case;
     end if;
  end if;
end process;

-- Seeleccionamos el origen de los cuadrantes en funcion del valor del registro base
pip_OrigenY <= KMitadY_240 when cuadrantePip(1)='1'  -- Cuadrantes 2 y 3 (abajo)
               else (others => '0');                 -- Cuadrantes 0 y 1 (arriba)
pip_OrigenX <= KMitadX_314 when cuadrantePip(0)='1'  -- Cuadrantes 1 y 3 (derecha)
               else (others => '0');                 -- Cuadrantes 0 y 2 (izquierda)

-- Componemos la direccion como corresponda (recolocacion y zoom x1/2)
pip_DireccionY <= dirRefrescoReg(18 downto 10) - pip_OrigenY;
pip_DireccionX <= dirRefrescoReg(9 downto 0) - pip_OrigenX;

-- Vemos si estamos dentro de las coordenadas de ese cuadrante o no
esZonaYPip <= '1' when (dirRefrescoReg(18 downto 10)>=pip_OrigenY and dirRefrescoReg(18
downto 10)<=(pip_OrigenY+KMitadY_240))
              else '0';
esZonaXPip <= '1' when (dirRefrescoReg(9 downto 0)>=pip_OrigenX and dirRefrescoReg(9
downto 0)<=(pip_OrigenX+KMitadX_314))
              else '0';
dentroZonaPip <= esZonaYpip and esZonaXpip;
```

```
-- Al componer la direccion de Pip hay que tener en cuenta que hay que reescalar (Zoom x1/2)
direccionPip <= pip_DireccionY(7 downto 0) & "0" & pip_DireccionX(8 downto 0) & "0" when
dentroZonaPip='1'
             else dirRefrescoBase;

--Asignaciones Concurrentes (hacia fuera del modulo)
esZonaPip <= dentroZonaPip;
dirPip <= direccionPip;
-------------------------------------------------------------
-----  Fin del Pip
-------------------------------------------------------------



-------------------------------------------------------------
-----  Inicio del Wipe
-------------------------------------------------------------
-- Registro con el tamanyo de la zona estatica del wipe
elProcesoWipe:
process (resetAjustes, cargaContadorWipe, vSync, tPeticion, tAjuste)
begin
  if (resetAjustes = '0') then
    contadorWipe <= KMitadX_314;
  elsif (vSync='1' and vSync'event) then
    if (tPeticion = tWIPE) then
      case (tAjuste) is
        when tMas =>
          contadorWipe <= contadorWipe + 32 ;
        when tMenos =>
          contadorWipe <= contadorWipe - 32 ;
        when others =>
          contadorWipe <= contadorWipe;
      end case;
    end if;
  end if;
end process;

-- Comprobamos si estamos en la zona estatica o no
esZonaWipe <= '1' when (dirRefrescoBase(9 downto 0)>=contadorWipe)
              else '0';
-------------------------------------------------------------
-----  Fin del Wipe
-------------------------------------------------------------
```

```
-------------------------------------------------------------
-----  Inicio del Zoom
-------------------------------------------------------------
-- Contador ciclico modulo 4 para seleccionar el factor del zoom
-- 00: x1/2, 01: normal; 10: x2; 11: x4
elModoZoom:
process(resetAjustes, vSync, tPeticion, tAjuste, contadorZoom)
  begin
  -- Si hay un reset cargamos por defecto el zoom x2
  if (resetAjustes='0') then
    contadorZoom<="10";
  elsif (vSync'event and vSync='1') then
    if (tPeticion=tZum) then
      case (tAjuste) is
        when tMas =>
          case (contadorZoom) is
            when "00" => contadorZoom <= "01";
            when "01" => contadorZoom <= "10";
            when "10" => contadorZoom <= "11";
            when "11" => contadorZoom <= "00";
            when others => contadorZoom<="10";
          end case;
        when tMenos =>
          case (contadorZoom) is
            when "00" => contadorZoom <= "11";
            when "11" => contadorZoom <= "10";
            when "10" => contadorZoom <= "01";
            when "01" => contadorZoom <= "00";
            when others => contadorZoom<="10";
          end case;
        when others =>
          null;
      end case;
    end if;
  end if;
end process;

-- Calculo de las direcciones posibles
zoomd2 <= dirRefrescoBase (17 downto 10) & "0" & dirRefrescoBase (8 downto 0) & "0";
zoomx2 <= "0" & dirRefrescoBase (18 downto 11) & "0" & dirRefrescoBase(9 downto 1);
zoomx4 <= "00" & dirRefrescoBase (18 downto 12) & "00" & dirRefrescoBase(9 downto 2);
```

```
-- Seleccionamos la que corresponda
muxZoom <= zoomd2 when contadorZoom="00"
          else dirRefrescoBase when contadorZoom="01"
          else zoomx2 when contadorZoom="10"
          else zoomx4 ;
dirZoom <= muxZoom;
-------------------------------------------------------------
-----  Fin del Zoom
-------------------------------------------------------------


-------------------------------------------------------------
-----  Inicio del Mosaico
-------------------------------------------------------------
-- Registro para almacenar la mascara que se utiliza al calcular la direccion de mosaico
-- Para ello separamos por un lado los bits que hacen referencia a las lineas y a los pixeles
-- Para aumentar, desplazar a la izquierda metiendo ceros
-- Para disminuir, desplazar a la derecha, metiendo unos
RegistroMascara:
process(resetAjustes, vSync, tPeticion, tAjuste, contadorZoom)
  begin
  if (resetAjustes='0') then
    -- Por defecto un tamanyo de 8
    regMascaraLinea <= "111111000";
    regMascaraPixel <= "1111111000";
  elsif (vSync'event and vSync='1') then
    if (tPeticion=tMosaico) then
      case (tAjuste) is
        when tMas =>
          regMascaraLinea <= regMascaraLinea(7 downto 0) &'0';
          regMascaraPixel <= regMascaraPixel( 8 downto 0) & '0';
        when tMenos =>
          regMascaraLinea <='1' & regMascaraLinea(8 downto 1) ;
          regMascaraPixel <= '1' & regMascaraPixel(9 downto 1) ;
        when others =>
          null;
      end case;
    end if;
  end if;
end process;

-- Calculamos la direccion de mosaico a partir del registro de mascara
dirMosaicoLinea <= regMascaraLinea and dirRefrescoBase(18 downto 10);
dirMosaicoPixel <= regMascaraPixel and dirRefrescoBase(9 downto 0);
dirMosaicoAux <= dirMosaicoLinea & dirMosaicoPixel;
dirMosaico <= dirMosaicoAux;

end Ajustes_Arquitectura;
```

interfazTecladoMaquinaEstados.vhd

```
----------------------------------------------------------------------------------------------------------
-- Interfaz entre el teclado y la maquina de efectos
----------------------------------------------------------------------------------------------------------
-- Entradas : dos relojes : el general (clock) y el reloj del teclado (ps2clk)
--        el reset del sistema
--        los datos que se reciben del teclado
--
-- Salida : el registro de estado, que indica el estado de la maquina de efectos en funcion
de la ultima tecla pulsada
--
--      0000 Cancel    0001 Still      0010 Recall       0011 Strobe
--      0100 Wipe      0101 Pip        0110 Zoom         0111 Mosaico
--      1000 Flip      1001 Rotate     1010 Strobe ciclico
--
-- Tambien devuelve el tipo de ajuste: SIN AJUSTE, POSITIVO (+) o NEGATIVO (-)
----------------------------------------------------------------------------------------------------------

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


--------------------------------------------------------------------------------
-- Declaracion de la entidad
--------------------------------------------------------------------------------
entity interfaz_teclado_maquina_de_estados is
   port (
      reset: in std_logic;
      clock: in std_logic;
      ps2Clk: in std_logic;                  -- Reloj del teclado
      ps2Data: in std_logic;                  -- Datos del teclado
      vsync : in std_logic;           -- La usaremos para cargar el registro cuando haya un flanco de
subida       registroEstado: out std_logic_vector (2 downto 0)   -- Devuelve el estado
      registroestado:out std_logic_vector (3 downto 0); -- Registro para almacenar el estado
      registroajustes:out std_logic_vector(1 downto 0)  -- Registro para almacenar el estado
   );
end interfaz_teclado_maquina_de_estados;
```

```
--------------------------------------------------------------------------------
-- Declaracion de la arquitectura
--------------------------------------------------------------------------------
architecture interfaz_teclado_maquina_de_estados_aquitectura of
interfaz_teclado_maquina_de_estados is

----------------------------------------------------------------------------------------------------
-- Declaracion de senyales
----------------------------------------------------------------------------------------------------
signal tecla: std_logic_vector(7 downto 0);        --Codigo de la tecla pulsada en el teclado
signal pulsacion :std_logic;            --Indica que ha habido una pulsacion
signal regEstadoAux:std_logic_vector (3 downto 0);      -- Registro auxiliar para guardar el estado
tras una pulsacion
signal regAjustesAux :std_logic_vector (1 downto 0);   -- Registro auxiliar de ajustes
signal resetear :std_logic;            -- Para resetear el registro interno de ajustes cuando hay un
flanco de subida de vsync

----------------------------------------------------------------------------------------------------
-- Declaracion de constantes
----------------------------------------------------------------------------------------------------

-- Constantes de estado
   constant CANCEL: std_logic_vector (3 downto 0):= "0000" ;
   constant STILL: std_logic_vector  (3 downto 0):= "0001" ;
   constant RECALL: std_logic_vector (3 downto 0):= "0010" ;
   constant STROBE: std_logic_vector (3 downto 0):= "0011" ;
   constant WIPE: std_logic_vector   (3 downto 0):= "0100" ;
   constant PIP: std_logic_vector    (3 downto 0):= "0101" ;
   constant ZOOM: std_logic_vector   (3 downto 0):= "0110" ;
   constant MOSAICO: std_logic_vector(3 downto 0):= "0111" ;
   constant FLIP: std_logic_vector   (3 downto 0):= "1000" ;
   constant ROTATE: std_logic_vector (3 downto 0):= "1001" ;
   constant STROBECICLICO: std_logic_vector(3 downto 0):= "1010" ;

-- Constantes de ajuste
   constant SINAJUSTE : std_logic_vector (1 downto 0):= "00";
   constant POSITIVO : std_logic_vector  (1 downto 0) := "01";
```

```
-------------------------------------
-- Declaracion del componente para comprobar las pulsaciones
-------------------------------------
component comprobador_tecla
  port (
    rst: IN std_logic;                    --Reset del sistema
    clk: IN std_logic;                    --Reloj del sistema
    ps2Clk: IN std_logic;                 --Reloj del teclado
    ps2Data: IN std_logic;                --Datos del teclado
    tecla: OUT std_logic_vector (7 downto 0); --Codigo de la tecla pulsada
    pulsacion: OUT std_logic              --Indica que se ha pulsado una tecla (activo a alta)
 );
end component;
-------------------------------------

begin

-------------------------------------
-- Instanciacion del componente para comprobar las pulsaciones
-------------------------------------
elComprobadorTeclas: comprobador_tecla
  port map (
    rst => reset,
    clk => clock,
    ps2Clk => ps2Clk,
    ps2Data => ps2Data,
    tecla => tecla,
    pulsacion => pulsacion
  );
-------------------------------------

-------------------------------------
-- Registro de Ajustes
-------------------------------------
-- Guarda el ajuste en el que se queda el registro tras llegar haber un flanco de subida en vsync
-- Depender del reloj para que resete solo durante un ciclo de reloj
-------------------------------------
elRegistroDeAjustes:
process (vsync, reset, regAjustesAux, clock)
begin
  if 8reset='0') then
    registroAjustes <= SINAJUSTE;
  elsif (vsync='1' and vsync'event) then
    registroAjustes <= regAjustesaux;
  end if;
end process;
```

```
-------------------------------------
-- Registro de Ajustes Auxiliar
-------------------------------------
-- Se carga cada vez que hay una pulsacion de una tecla de ajuste.
-- Tras un vSync se reseteara para evitar que se lea mas de una vez su valor
-------------------------------------
elRegustroAjustesAux:
process (clock, reset, tecla, pulsacion, resetear,regAjustesAux)
begin
  if (reset = '0') then                  -- El reset a cero inicializa el valor del registro a CANCEL
    regAjustesAux <= SINAJUSTE;
    resetear <='0';
  elsif (clock'event and clock = '1') then -- Solo responde ante flanco de subida del reloj
    -- En primer lugar vemos si tenemos que resetear
    -- Siempre que haya una pulsacion la atenderemos, es prioritaria sobre el reseteo
    -- Si se diera la casualidad de que reseteamos justo cuando hay una pulsacion, el valor del
registro interno ser el valor de la pulsacin si fuese + o -
    if (vsync='0') then
      resetear <='0';
    elsif vsync='1' and resetear='0' then
      resetear <='1';
      regAjustesAux<= SINAJUSTE;
    end if;
    if (pulsacion ='1') then             -- Dependiendo de la tecla pulsada, incrementamos o
decrementamos
      case tecla is
        when "01111001" => regAjustesAux<= POSITIVO;  --Tecla "+" -> codigo = 79. La
usaremos para positivo
        when "01111011" => regAjustesAux<= NEGATIVO;  --Tecla "-" -> codigo = 7B. La
usaremos para negativo
        when others     =>  null;
      end case;
    end if;
  end if;
end process;


-------------------------------------
-- Registro de Estado
-------------------------------------
-- Guarda el estado en el que se queda el registro tras llegar haber un flanco de subida en vsync
(cuando termina un frame)
-------------------------------------
elRegistroDelEstado:
process (vsync, reset, regEstadoAux)
begin
```

```
  if reset='0' then
    registroestado <= (others =>'0');
  elsif vsync'event and vsync='1' then
    registroestado <= regEstadoaux;
  end if;
end process;


-------------------------------------
-- Registro de Estado Auxiliar
-------------------------------------
-- Carga el valor del estado que se corresponde con la tecla pulsada.
-- No se hara "efectivo" hasta el siguiente vSync, cuando pase al registro de salida
-------------------------------------
elRegistroDeEstadoAux:
process (clock, reset, tecla, pulsacion)
begin
  if (reset = '0') then   -- El reset a cero inicializa el valor del registro a CANCEL
    regEstadoAux<= CANCEL;
  elsif (clock'event and clock = '1') then -- Solo responde ante flanco de subida del reloj
    if (pulsacion ='1') then              -- Dependiendo de la tecla pulsada, incrementamos o
decrementamos
      case tecla is
        when "00011011" => regEstadoAux<= STILL;      -- "S" -> codigo = 1B   pausa
        when "00101101" => regEstadoAux<= RECALL;  -- "R" -> codigo = 2D  memoria
        when "00100100" => regEstadoAux<= STROBE; -- "E" -> codigo =24.   estrobe
        when "01001101" => regEstadoAux<= PIP;          -- "P" -> codigo =4D.   Pip
        when "00011101" => regEstadoAux<= WIPE;       -- "W" -> codigo =1D.  Wipe
        when "00011010" => regEstadoAux<= ZOOM;     -- "Z" -> codigo =1A.    Zoom
        when "00111010" => regEstadoAux<= MOSAICO; -- "M" -> codigo =3A. Mosaico
        when "00100001" => regEstadoAux<= CANCEL;   -- "C" -> codigo =21   cancelar
        when "00101011" => regEstadoAux<= FLIP;          -- "F" -> codigo =2B   reflejo
        when "00101010" => regEstadoAux<= ROTATE;   -- "V" -> codigo =2A. rotacion
        when "00110101" => regEstadoAux<= STROBECICLICO; -- "Y" -> =35 estrobe ciclico
        when others     => null;
      end case;
    end if;
  end if;
end process;
-------------------------------------

end interfaz_teclado_maquina_de_estados_aquitectura;
```

comprobadorTecla.vhd

```
--------------------------------------------------------------------------------
-- MODULO QUE ACTUA SOBRE EL INTERFAZ DEL TECLADO
--------------------------------------------------------------------------------
-- ENTIDAD: comprobador_tecla
-- CONEXIONES:
-- PUERTOS ENTRADA:
--    clk: Entrada del reloj
--    rst: Reset del circuito (activo a baja)
--    ps2Clk: Reloj del teclado
--    ps2Data: Entrada de datos del teclado
-- PUERTOS SALIDA:
--    tecla(7..0): Codigo de la tecla que se ha pulsado
--    pulsacion: Indica que se ha pulsado una tecla (activa a alta)
--------------------------------------------------------------------------------
-- Este modulo actua por encima del modulo que hace de interfaz con el teclado PS2
-- Dadas las senyales de reset, reloj, reloj del teclado y un dato del teclado devuelve dos
senyales:
--    pulsacion -> indica que ha habido una pulsación de una tecla cuando ya
--                 se ha dejado de pulsar dicha tecla (activa a alta)
--    tecla -> devuelve el codigo de la tecla pulsada (codigo de 8 bits)
--------------------------------------------------------------------------------


library IEEE;
use IEEE.std_logic_1164.ALL;


--------------------------------------------------------------------------------
-- Declaracion de la entidad
--------------------------------------------------------------------------------
entity comprobador_tecla is
   port (
      rst: IN std_logic;                    --Reset del sistema
      clk: IN std_logic;                    --Reloj del sistema
      ps2Clk: IN std_logic;                 --Reloj del teclado
      ps2Data: IN std_logic;                --Datos del teclado
      tecla: OUT std_logic_vector (7 downto 0); --Codigo de la tecla pulsada
      pulsacion: OUT std_logic              --Indica que se ha pulsado una tecla (activo a alta)
   );
end comprobador_tecla;
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
-- Declaracion de la arquitectura
--------------------------------------------------------------------------------
architecture comprobador_tecla_Arquitectura OF comprobador_tecla is


--------------------------------------------------------------------------------
-- Declaracion del componente del interfaz con el tecaldo
--------------------------------------------------------------------------------
component interfaz_teclado_PS2
   port (
      clk: IN std_logic;                    --Reloj del sistema
      rst: IN std_logic;                    --Reset del sistema (activo a baja)
      ps2Clk: IN std_logic;                 --Reloj del teclado
      ps2Data: IN std_logic;                --Datos de entrada del teclado
      data: OUT std_logic_vector (7 downto 0); --Datos de salida del controlador
      newData: OUT std_logic;               --Pone a uno cuando se manda un dato y se pone a
esperar confirmacion
      newDataAck: IN std_logic              --Espera a que le llegue un uno como confirmacion
   );
end component;
--------------------------------------------------------------------------------

   type tEstado is (esperandoNuevoDato, compruebaF0, esperandoNuevoDato2,
acknowledgeNuevoDato);
   signal estado: tEstado;

   signal data: std_logic_vector (7 downto 0); --Codigo que viene del interfaz con el teclado
   signal newData: std_logic;                --Indica que hay un nuevo dato (activo a alta)
   signal newDataAck: std_logic;             --Confirma la recepcion de un dato (activo a alta)
   signal cargaCodigoTecla: std_logic;       --Indica que se muestre el valor de la tecla (activo a
alta)


begin
```

```
--------------------------------------------------------------------------------
-- Instanciacion del interfaz con el teclado
--------------------------------------------------------------------------------
interfazConTeclado: interfaz_teclado_PS2
   port map (clk, rst, ps2Clk, ps2Data, data, newData, newDataAck);
--------------------------------------------------------------------------------



--------------------------------------------------------------------------------
-- Maquina de estados del controlador
--------------------------------------------------------------------------------
-- La maquina va a disponer de 4 estados para llevar a cabo todo el control, que son los
siguientes
--   esperandoNuevoDato ->esta en este estado antes de recibir un dato
--   compruebaF0 ->comprueba si el codigo recibido es F0
--   esperandoNuevoDato2 -> si el codigo recibido es F0, entonces espera el codigo de depresion
--   acknowledgeNuevoDato -> estado de confirmacion de tecla recibida
--------------------------------------------------------------------------------



--------------------------------------------------------------------------------
-- Generacion de las senyales tipo Moore
--------------------------------------------------------------------------------
laGeneracionDeSenyales:
process (estado,data)
begin
    newDataAck <= '0';
    cargaCodigoTecla <= '0';

    case estado is

       when compruebaF0 =>
          newDataAck <= '1';        --Si estamos esperando un "F0", indicamos confirmacion de
recepcion

       when acknowledgeNuevoDato =>
          newDataAck <= '1';        --Si hemos recibido el "F0" y otro dato, indicamos tambien el ack
          cargaCodigoTecla <='1';
          tecla<=data;

       when others => null;         --En otros casos (esperandoNuevoDato y esperandoNuevoDato2)
nada (seguira valiendo cero)

    end case;

 end process;
```

```
 ------------------------------------
-- Generacion del estado siguiente:
------------------------------------
generacionEstadoSiguente:
process (rst, clk, newdata,data)
begin
   if (rst='0') then
      estado <= esperandoNuevoDato;
    elsif (clk'EVENT and clk='1') then

      case estado is

        when esperandoNuevoDato =>
          --Si estamos esperando un dato y llega uno, pasamos a ver si es un "F0" (tecla soltada)
          if (newData='1') then
             estado <= compruebaF0;
          end if;

        when compruebaF0 =>
          --Estamos esperando que nos llegue un "F0" (codigo de depresion)
          if (data="11110000") then
             --Si el dato que nos llega es un "F0", pasamos a esperar el siguiente dato
             estado <= esperandoNuevoDato2;
          else    --Si no, volvemos al primer estado
             estado <= esperandoNuevoDato;
          end if;

        when esperandoNuevoDato2 =>
          --Estamos esperando un dato (tecla pulsada)
          if (newData='1') then   --Si llega ese dato, pasamos al estado de confirmacion
             estado <= acknowledgeNuevoDato;
          end if;

        when others =>
          estado <= esperandoNuevoDato;
      end case;
   end if;
end process;

----------------------------------------
-- Asignacion continua de la senyal de carga
----------------------------------------
pulsacion <= cargaCodigoTecla;
----------------------------------------

end comprobador_tecla_Arquitectura;
```

interfazTecladoPS2.vhd

```
--------------------------------------------------------------------------------
-- INTERFAZ CON UN TECLADO PS2
--------------------------------------------------------------------------------
-- ENTIDAD: interfaz_teclado_PS2
-- CONEXIONES:
-- PUERTOS ENTRADA:
--    clk: Entrada del reloj
--    rst: Reset del circuito (activo a baja)
--    ps2Clk: Reloj del teclado
--    ps2Data: Entrada de datos del teclado
--    newDataAck: Senyal de confirmacion de que se ha recibido el dato
-- PUERTOS SALIDA:
--    data(7..0): Codigo de la tecla que se ha pulsado
--    newData: Indica que se ha pulsado una tecla (activa a alta)
--------------------------------------------------------------------------------
-- Implementa un interfaz on un teclado tipo PS2
-- Recibe del teclado su reloj y una senyal de datos.
-- Como la comunicacion con el teclado es mediante un unica linea, tiene que ir
--  componiendo el codigo de la tecla pulsada para poder suministrarlo con 8 bits.
-- Ademas, dispone de una senyal para indicar que se ha producido una nueva
--  pulsacion para poder informar al resto de componentes, y una senyal para que
--  se le indique que ya se muestreo el valor de la pulsacion anterior.
-- Para evitar problemas con los rebotes y con las diferencias de los relojes,
--  sincronizamos el reloj del teclado con el de entrada al modulo.
--------------------------------------------------------------------------------


library IEEE;
use IEEE.std_logic_1164.ALL;


--------------------------------------------------------------------------------
-- Declaracion de la entidad
--------------------------------------------------------------------------------
entity interfaz_teclado_PS2 IS
   port (
      clk: IN std_logic;                    --Reloj del sistema
      rst: IN std_logic;                    --Reset del sistema (activo a baja)
      ps2Clk: IN std_logic;                    --Reloj del teclado
      ps2Data: IN std_logic;                    --Datos de entrada del teclado
      data: OUT std_logic_vector (7 downto 0); --Datos de salida del controlador
      newData: OUT std_logic;                --Pone a uno cuando se manda un dato y se pone a
esperar confirmacion
      newDataAck: IN std_logic               --Espera a que le llegue un uno como confirmacion
   );
end interfaz_teclado_PS2;
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
-- Declaracion de la arquitectura
--------------------------------------------------------------------------------
architecture interfaz_teclado_PS2_Arquitectura OF interfaz_teclado_PS2 IS

  type tEstado is (esperandoDato, esperandoNuevoAck);
  signal estado: tEstado;   --indica el estado en el que se encuentra el controlador

  signal sacaTecla: std_logic;               --Senyal para sacar el valor de la tecla (activo a alta)
  signal datoValido: std_logic;              --Indica que hay un dato valido (activo a alta)
  signal ultimoBitRecibido: std_logic;       --Indica que se recibio el ultimo bit del teclado (alta)
  signal relojTecladoSincronizado: std_logic; --Senyal del teclado sincronizada
  signal relojTecladoFlanco: std_logic;      --Flanco de bajada del reloj del teclado (alta)
  signal paridadCorrecta: std_logic;         --Indica que la paridad impar es correcta (baja)
  signal registroSalida: std_logic_vector(10 downto 0);  --Registro intermedio

begin


-------------------------------------
-- Sincronizador
-------------------------------------
-- Sincroniza el reloj del sistema con el del teclado
-- (Equivale a dos biestables conectados en cascada y latcheados por el reloj del sistema)
-------------------------------------
sincronizador:
process (rst, clk)
   variable aux1: std_logic;            --Valor de la senyal hace un ciclo
begin
   if (rst='0') then
      aux1 := '1';
      relojTecladoSincronizado <= '1';
   elsif (clk'EVENT and clk='1') then
      -- Ante un flanco se subida del reloj, las senyales pasan al siguiente biestable
      relojTecladoSincronizado <= aux1;    --ps2Clk -> aux1 -> relojTecladoSincronizado
```

```
      aux1 := ps2Clk;
    end if;
end process;

-------------------------------------
-- Detector de flancos en el reloj del teclado
-------------------------------------
-- Comparamos lo que valia la senyal hace un ciclo y hace dos y vemos si hay flancos
-------------------------------------
detectorFlancos:
process (rst, clk)
   variable aux1, aux2: std_logic;          --Senyales hace uno y dos ciclos respectivamente
begin
   --Vale 1 cuando tengamos: un 0 hace un ciclo y un 1 hace dos ciclos (flanco bajada)
   if (rst='0') then
      relojTecladoFlanco <= '0';                -- Vale 1 con flanco de bajada
      aux1 := '1';
      aux2 := '1';
   elsif (clk'EVENT and clk='1') then
      relojTecladoFlanco <= (not aux1) and aux2; -- Vale 1 con flanco de bajada
      aux2 := aux1;                          --   relojTecladoSincronizado -> aux1 -> aux2
      aux1 := relojTecladoSincronizado;          --Ante un flanco de subida del reloj, las senyales
pasan al siguiente biestable:
   end if;
end process detectorFlancos;

-------------------------------------
-- Registro de desplazamiento
-------------------------------------
-- Con un reset o cuando se reciba el ultimo bit de la transmision, se incializa todo a uno
-- Si no, desplaza un lugar a la derecha para ir componiendo el codigo de la tecla
-------------------------------------
elRegistroDeDesplazamiento:
process (rst, clk)
begin
   if (rst='0') then
      registroSalida <= (others =>'1');
   elsif (clk'EVENT and clk='1') then
      if (ultimoBitRecibido='1') then       --Cuando haya recibido el ultimo bit, inicializa todo a unos
        registroSalida <= (others=>'1');
      elsif (relojTecladoFlanco='1') then   --Si todavia no ha llegado el ultimo bit, con un flanco de
bajada,
        registroSalida <= ps2Data & registroSalida(10 downto 1);  -- desplaza un lugar a la derecha
      end if;
    end if;
end process;

-------------------------------------
-- Paridad Impar
-------------------------------------
-- Comprueba si es correcta la paridad impar (numero de unos ha de ser impar)
-- (or exclusiva entre el dato (registroSalida(9..2)) y el bit de paridad (registroSalida(1)))
-- (Valdra 1 cuando el numero de unos total sea par)
-------------------------------------
paridadCorrecta <=
   ((registroSalida(9) xor registroSalida(8)) xor (registroSalida(7) xor registroSalida(6)))
     xor ((registroSalida(5) xor registroSalida(4)) xor (registroSalida(3) xor registroSalida(2)))
     xor registroSalida(1);


-------------------------------------
-- Ultimo Bit Recibido
-------------------------------------
-- Indica que hemos recibido el ultimo bit de una transmision
-- Vale 0 cuando reg(0) es el bit de Stop y 1 cuando reg(0) es el bit de Start
-------------------------------------
ultimoBitRecibido <= not registroSalida(0);
-------------------------------------



-------------------------------------
-- Dato Valido
-------------------------------------
-- Indica que tenemos un dato valido
-- Vale 1 cuando tenemos en reg(0) el bit de start y una paridad correcta (a uno)
-------------------------------------
datoValido <= ultimoBitRecibido and paridadCorrecta;
-------------------------------------



-------------------------------------
-- Regitro que almacena el codigo de la tecla
-------------------------------------
-- Sacaremos por la salida del controlador el valor de la tecla en los flancos de subida,cuando la
senyal sacaTecla valga uno.
-------------------------------------
elRegistroDelCodigoDeTecla:
process (rst, clk)
   begin
      if (rst='0') then
        data <= (others=>'0');
      elsif (clk'EVENT and clk='1') then    --Con un flanco de subida, sacamos del interfaz el
contenido del registro de datos
```

```
      if (sacaTecla='1') then
        data <= registroSalida(8 downto 1);
      end if;
    end if;
end process;


-------------------------------------
-- Maquina de estados del interfaz
-------------------------------------
--  Va a haber dos estados: esperandoDato y esperandoDatoack.
--  esperandoDato : antes de haber recibido un dato del teclado
--  esperandoDatoAck: todavia no ha recibido confirmacion de la recepcion de la tecla
-------------------------------------


-------------------------------------
-- Proceso para generar las senyales tipo Moore
-------------------------------------
generacionSenyalesMoore:
process (datoValido, estado)
begin
   sacaTecla <= '0';
   newData <= '0';

   case estado is

      when esperandoDato =>        --Si estamos esperando un dato y tenemos uno valido,
indicamos que se saque el que hay
         if (datoValido='1') then
           sacaTecla <= '1';
         end if;

      when esperandoNuevoAck =>    --Si esperamos una confirmacion, indicamos que queremos
un nuevo dato
         newData <= '1';

      when others => null;
   end case;
end process;
```

```
-------------------------------------
-- Proceso para generar el estado siguiente
-------------------------------------
generacionEstadoSiguiente:
process (datoValido, rst, clk)
begin
   if (rst='0') then
      estado <= esperandoDato;
   elsif (clk'event and clk='1') then

      case estado is

    when esperandoDato =>
       --Si estamos esperando un dato y tenemos uno valido, pasamos a esperar confirmacion
          if (datoValido='1') then
             estado <= esperandoNuevoAck;
           end if;

   when esperandoNuevoAck =>
          --Si estamos esperando confirmacion y la recibimos, paramos a esperar un nuevo dato
          if (newDataAck='1') then
             estado <= esperandoDato;
          end if;

   when others => null;
      end case;
   end if;
end process ;

end interfaz_teclado_PS2_Arquitectura;
```

programaRamDac.vhd

```
----------------------------------------------------------------------------
-- MODULO SUPERIOR PARA LA PROGRAMACION DEL RAMDAC EN TRUE COLOR O
GRISES
----------------------------------------------------------------------------
-- ENTIDAD: programaRamDac
-- CONEXIONES:
-- PUERTOS ENTRADA:
--    relojProgramador: Entrada del reloj a 50MHz
--    resetProgramador: Reset del circuito (activo a baja)
--    inicioProgramacion: Para que comience a programar (activo a alta)
-- PUERTOS SALIDA:
--    finProgramacion: Indica que ha terminado la programación (activo a alta)
--    WrPaleta: Senyal de write de la paleta del RamDac (activa a baja)
--    RdPaleta: Senyal de lectura de la paleta del RamDac (activa a baja)
-- PUERTOS ENTRADA/SALIDA:
--    RS(2..0): Seleccion de registro de la paleta (Register Select)
--    datosPaleta(7..0): Linea bidireccional con la paleta del RamDac
----------------------------------------------------------------------------
-- El modulo actua por encima del autentico programador y nos permite
--  programar el RamDac para que funcione con color real en el formato 5:5:5
--  funcionando con un solo flanco (single-edge) o en escala de grises, dependiendo
--  del "component" que hayamos "descomentado".
-- El hacerlo asi es simplemente por la facilidad de poder usar uno u otro metodo
--  cambiando unicamente una serie de comentarios.
-- Basicamente, lo que hace es permanecer en un estado inicial hasta que llegue
--  la senyal de inicio. Entonces, le indica al modulo interior que programe el
--  RamDac como corresponda y se pone a esperar a que este le indique que ha
--  terminado. Una vez hecho esto, pasa a un estado final e indica que ya se ha
--  programado correctamente.
----------------------------------------------------------------------------

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

----------------------------------------------------------------------------
-- Declaracion de la entidad
----------------------------------------------------------------------------
entity programaRamDac is
    port (
        relojProgramador: in STD_LOGIC;   -- Reloj de entrada (a 50MHz)
        resetProgramador: in STD_LOGIC;   -- Reset del circuito (activo a baja)
        inicioProgramacion: in STD_LOGIC; -- Senyal de inicio (activa a alta)
        finProgramacion: out STD_LOGIC;   -- Senyal de finalizacion (activa a alta)
        WrPaleta: out STD_LOGIC;          -- Senyal de escritura en la paleta (activa a baja)
        RdPaleta: out STD_LOGIC;          -- Senyal de lectura de la paleta (activa a baja)
        RS: inout STD_LOGIC_VECTOR (2 downto 0); -- Lineas de seleccion de registro (register
select)
        datosPaleta: inout STD_LOGIC_VECTOR (7 downto 0)   -- Linea bidireccional de
datosPaleta con el RamDac
    );
end programaRamDac;
----------------------------------------------------------------------------



----------------------------------------------------------------------------
-- Declaracion de la Arquitectura
----------------------------------------------------------------------------
architecture programaRamDac_Arquitectura of programaRamDac is



----------------------------------------------------------------------------
-- Programador en TrueColor con formato 5:5:5 y Single Edge
----------------------------------------------------------------------------
-- Comentar estas lineas y descomentar las del component siguiente para
--  hacerlo en escala de grises
----------------------------------------------------------------------------
component programaRamDacTrueColor is
    port (
        relojProgramador: in STD_LOGIC;   -- Reloj de entrada (a 50MHz)
        resetProgramador: in STD_LOGIC;   -- Reset del circuito (activo a baja)
        inicioProgramacion: in STD_LOGIC; -- Senyal de inicio (activa a alta)
        finProgramacion: out STD_LOGIC;   -- Senyal de finalizacion (activa a alta)
        WrPaleta: out STD_LOGIC;          -- Senyal de escritura en la paleta (activa a baja)
        RdPaleta: out STD_LOGIC;          -- Senyal de lectura de la paleta (activa a baja)
        RS: inout STD_LOGIC_VECTOR (2 downto 0); -- Lineas de seleccion de registro (register
select)
        datosPaleta: inout STD_LOGIC_VECTOR (7 downto 0)   -- Linea bidireccional de
datosPaleta con el RamDac
    );
end component;
----------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
-- Programador en escala de grises
--------------------------------------------------------------------------------
-- Comentar estas lineas y descomentar las del component anterior para
--  hacerlo en True Color 5:5:5 single edge
--------------------------------------------------------------------------------
--component programaRamDacGrises is
--    port (
--        relojProgramador: in STD_LOGIC;   -- Reloj de entrada (a 50MHz)
--        resetProgramador: in STD_LOGIC;   -- Reset del circuito (activo a baja)
--        inicioProgramacion: in STD_LOGIC; -- Senyal de inicio (activa a alta)
--        finProgramacion: out STD_LOGIC;   -- Senyal de finalizacion (activa a alta)
--        WrPaleta: out STD_LOGIC;          -- Senyal de escritura en la paleta (activa a baja)
--        RdPaleta: out STD_LOGIC;          -- Senyal de lectura de la paleta (activa a baja)
--        RS: inout STD_LOGIC_VECTOR (2 downto 0); -- Lineas de seleccion de registro (register
select)
--        datosPaleta: inout STD_LOGIC_VECTOR (7 downto 0)   -- Linea bidireccional de
datosPaleta con el RamDac
--    );
--end component;
--------------------------------------------------------------------------------


-- Estados de la maquina
   type TESTADO is (stInicial, stEspera1, stEspera2, stEspera3, stEspera4, stProgramando,
stTerminado);
   signal estadoActual: TESTADO;

-- Senyales auxiliares internas
   signal inicioProgramacionAux: std_logic;
   signal finProgramacionAux: std_logic;


begin


--------------------------------------------------------------------------------
-- Programador en TrueColor con formato 5:5:5 y Single Edge
--------------------------------------------------------------------------------
-- Comentar estas lineas y descomentar las del component siguiente para
--  hacerlo en escala de grises
--------------------------------------------------------------------------------
programaRamDacEnTrueColor: programaRamDacTrueColor
    port map (
        relojProgramador => relojProgramador,
        resetProgramador => resetProgramador,
        inicioProgramacion => inicioProgramacionAux,
        finProgramacion => finProgramacionAux,
        WrPaleta => WrPaleta,
        RdPaleta => RdPaleta,
        RS => RS,
        datosPaleta => datosPaleta
    );
--------------------------------------------------------------------------------



--------------------------------------------------------------------------------
-- Programador en escala de grises
--------------------------------------------------------------------------------
-- Comentar estas lineas y descomentar las del component anterior para
--  hacerlo en True Color 5:5:5 single edge
--------------------------------------------------------------------------------
--programaRamDacEnEscalaDeGrises: programaRamDacGrises
--    port map (
--        relojProgramador => relojProgramador,
--        resetProgramador => resetProgramador,
--        inicioProgramacion => inicioProgramacionAux,
--        finProgramacion => finProgramacionAux,
--        WrPaleta => WrPaleta,
--        RdPaleta => RdPaleta,
--        RS => RS,
--        datosPaleta => datosPaleta
--    );
--------------------------------------------------------------------------------



-------------------------------------
-- Maquina de estados para controlar el proceso de programacion
-- Es necesaria porque necesitamos relajar los tiempos de las senyales al tener
--  un reloj de 50MHz
-- Realmente lo unico que, cuando le llega la senyal de inicio, se la pasa al
--  programador, se la mantiene durante unos ciclos y despues se pone a esperar
--  a que el programador le indique que ha terminado.
-------------------------------------
process(relojProgramador, resetProgramador)
begin
   if (resetProgramador = '0') then
      estadoActual <= stInicial;
   elsif (relojProgramador'event AND relojProgramador = '1') then
      case estadoActual is
```

```
    -- Nos mantenemos en el estado inicial mientras no nos llegue la senyal de inicio
      when stInicial =>
        if (inicioProgramacion = '0') then
          estadoActual <= stInicial;
        else
          estadoActual <= stEspera1;
        end if;
      -- Entramos en una serie de ciclos que unicamente mantienen la senyal durante mas
tiempo
      when stEspera1 =>
        estadoActual <= stEspera2;
      when stEspera2 =>
        estadoActual <= stEspera3;
      when stEspera3 =>
        estadoActual <= stEspera4;
      when stEspera4 =>
        estadoActual <= stProgramando;
      -- Permanecemos en este estado hasta que se termine de programar
      when stProgramando =>
        if (finProgramacionAux = '0') then
          estadoActual <= stProgramando;
        else
          estadoActual <= stTerminado;
        end if;
      -- Estado final; hemos terminado de programar el RamDac
      when stTerminado =>
        estadoActual <= stTerminado;
    end case;
  end if;
end process;
-------------------------------------


-------------------------------------
-- Proceso para dar las senyales que corresponan en funcion del estado en el que nos
encontremos
-- En los de espera se pone a uno la senyal de inicio de programacion y en los demas a cero
-- La senyal de fin, solo se pone a uno cuando hemos llegado al ultimo estado
-------------------------------------
process(estadoActual)
begin
  finProgramacion <= '0';
  case estadoActual is
    -- En el estado inicial estamos esperando a que nos den la senyal de inicio
    when stInicial =>
      inicioProgramacionAux <= '0';
    -- Tenemos que mantener la senyal activa durante unos ciclos
    when stEspera1 =>
      inicioProgramacionAux <= '1';
    when stEspera2 =>
      inicioProgramacionAux <= '1';
    when stEspera3 =>
      inicioProgramacionAux <= '1';
    when stEspera4 =>
      inicioProgramacionAux <= '1';
    -- Estamos esperando mientras se esta programando el RamDac
    when stProgramando =>
      inicioProgramacionAux <= '0';
    -- Ya hemos terminado, asi que activamos la senyal de fin
    when stTerminado =>
      inicioProgramacionAux <= '0';
      finProgramacion <= '1';
  end case;
end process;
-------------------------------------


end programaRamDac_Arquitectura;
```

programaRamDacTrueColor.vhd

```
--------------------------------------------------------------------------------
-- PROGRAMACION DEL RAMDAC EN C0LOR REAL CON FORMATO 5:5:5 y "single edge"
--------------------------------------------------------------------------------
-- ENTIDAD: programaRamDacTrueColor
-- CONEXIONES:
-- PUERTOS ENTRADA:
--    relojProgramador: Entrada del reloj a 50MHz
--    resetProgramador: Reset del circuito (activo a baja)
--    inicioProgramacion: Para que comience a programar (activo a alta)
-- PUERTOS SALIDA:
--    finProgramacion: Indica que ha terminado la programación (activo a alta)
--    WrPaleta: Senyal de write de la paleta del RamDac (activa a baja)
--    RdPaleta: Senyal de lectura de la paleta del RamDac (activa a baja)
-- PUERTOS ENTRADA/SALIDA:
--    RS(2..0): Seleccion de registro de la paleta (Register Select)
--    datosPaleta(7..0): Linea bidireccional con la paleta del RamDac
--------------------------------------------------------------------------------
-- El modulo programa el RamDac para que funcione con color real en el formato 5:5:5
--  funcionando con un solo flanco (single-edge)
-- Simplemente tiene que configurar adecuadamente el redistro de Comandos A
-- Para comenzar a programar ponemos a uno la senyal de "inicioProgramacion" y
--  esperamos hasta que se ponga en alta la senyal "finProgramacion".
-- Notas:
-- - Para evitar problemas con la temporizacion, como este modulo solo se ejecuta
--   una vez al principio, utilizamos internamente un reloj relajado para evitar
--   problemas de temporizacion a la hora de programarlo.
-- - Los valores de "RS" y "datosPaleta" se han colocado en dos arrays para permitir
--   facilmente modificaciones en la configuracion, de ahi que se hayan mantenido
--   con cinco posiciones, que en principio es lo maximo que nos haria falta. Ahora
--   mismo, como solo nos hacen falta tres, repetimos durante los primeros los
--   mismos valores (como si no hicieramos nada).
--------------------------------------------------------------------------------


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

--------------------------------------------------------------------------------
-- Declaracion de la entidad
--------------------------------------------------------------------------------
entity programaRamDacTrueColor is
    port (
        relojProgramador: in STD_LOGIC;   -- Reloj de entrada (a 50MHz)
        resetProgramador: in STD_LOGIC;   -- Reset del circuito (activo a baja)
        inicioProgramacion: in STD_LOGIC; -- Senyal de inicio (activa a alta)
        finProgramacion: out STD_LOGIC;   -- Senyal de finalizacion (activa a alta)
        WrPaleta: out STD_LOGIC;          -- Senyal de escritura en la paleta (activa a baja)
        RdPaleta: out STD_LOGIC;          -- Senyal de lectura de la paleta (activa a baja)
        RS: inout STD_LOGIC_VECTOR (2 downto 0); -- Lineas de seleccion de registro (register
select)
        datosPaleta: inout STD_LOGIC_VECTOR (7 downto 0)   -- Linea bidireccional de
datosPaleta con el RamDac
    );
end programaRamDacTrueColor;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- Declaracion de la arquitectura
--------------------------------------------------------------------------------
architecture programaRamDacTrueColor_Arquitectura of programaRamDacTrueColor is


-- Estados para la máquina de mealy
  type TESTADO is (stInicial, stEscribe, stCicloEscritura, stSiguienteEscribe);
  signal estadoActual: TESTADO;
  signal estadoSiguiente: TESTADO;

-- Tipos para los arrays de inicializacion
  type TARRAYDAC is array (0 to 5) of STD_LOGIC_VECTOR (7 downto 0);
  type TARRAYRS is array (0 to 5) of STD_LOGIC_VECTOR (2 downto 0);

  -- Unicamente necesitamos colocar el valor "10100000" en el registro de comando A
  constant initDAC: TARRAYDAC := (
    "10100000",        -- Registro de Comando A en TrueColor
    "10100000",        -- Registro de Comando A en TrueColor
    "10100000",        -- Registro de Comando A en TrueColor
    "10100000",        -- Registro de Comando A en TrueColor
    "10100000",        -- Registro de Comando A en TrueColor
    "10100000"        -- Registro de Comando A en TrueColor
    );
```

```
   -- Unicamente accederemos al registro de comando A por lo que vale siempre "110"
  constant initRS: TARRAYRS := (
    "110",            -- Registro de Comando A
    "110",            -- Registro de Comando A
    "110",            -- Registro de Comando A
    "110",            -- Registro de Comando A
    "110",            -- Registro de Comando A
    "110"            -- Registro de Comando A
    );

-- Senyales para acceder a las distintas posiciones de los arrays de incializacion
  signal indiceInit: INTEGER range 0 to 5; -- Indice para acceder
  signal incrementaIndiceInit: STD_LOGIC;  -- Si vale uno, se incrementara el indice

-- Senyales para crear un reloj de 12'5MHz a partir del de entrada de 50MHz
  signal reloj125MHz: STD_LOGIC;
  signal contadorGray: STD_LOGIC_VECTOR (1 downto 0);

-- Senyales para usar las senyales "datosPaleta" y "RS" como buffers triestado
--  (hay que hacerlo asi porque comparten lineas con la de ethernet)
  signal progDatosPaleta: STD_LOGIC_VECTOR (7 downto 0); -- Bidireccional con la paleta
  signal progRS: STD_LOGIC_VECTOR (2 downto 0);          -- Hacia el registro RS
  signal latchDatosPaleta: STD_LOGIC; -- Registro para mantener estables los datos de la paleta
  signal latchRS: STD_LOGIC;          -- Resgistro para mantener estables los valores del registro
RS


begin


-------------------------------------
-- Divisor de frecuencia para obtener el reloj de 12'5MHz a partir del de 50 que entra
-- Usa un contador gray para minimizar la logica necesaria
-------------------------------------
divisorDeFrecuencia:
process (resetProgramador, relojProgramador)
begin
  if resetProgramador = '0' then
    contadorGray <= "00";
  elsif (relojProgramador'EVENT and relojProgramador = '1') then
    case (contadorGray) is
      when "00" => contadorGray <= "01";
      when "01" => contadorGray <= "11";
      when "11" => contadorGray <= "10";
      when "10" => contadorGray <= "00";
      when others => contadorGray <= "00";
    end case;
  end if;
end process;
-------------------------------------



-------------------------------------
-- La senyal de reloj, la asignamos constantemente
-------------------------------------
reloj125MHz <= contadorGray(1);
-------------------------------------



-------------------------------------
-- Como no vamos a leer del RamDac, desactivamos las lecturas (siempre a uno)
-------------------------------------
RdPaleta <= '1';
-------------------------------------



-------------------------------------
-- Proceso principal sincronizado con el reloj de 12'5MHz
-- Es el el encargado de hacer los cambios de estado y sincronizar los cambios de senyales
-------------------------------------
procesoPrincipalClockeado:
process (resetProgramador, reloj125MHz)
begin
  if resetProgramador = '0' then
    --Inicializamos los valores
    estadoActual <= stInicial;
    indiceInit <= 0;
  elsif (reloj125MHz'event and reloj125MHz = '1') then
    estadoActual <= estadoSiguiente;

    -- Incrementamos el indice de acceso a los arrays de configuracion
    if (incrementaIndiceInit = '1') then
      if (indiceInit < 5) then
        indiceInit <= indiceInit + 1;
      else
        indiceInit <= 0;
      end if;
    end if;

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

```
-------------------------------------
-- Maquina de estados principal
-------------------------------------
maquinaEstadosPrincipal:
process (estadoActual, inicioProgramacion, indiceInit)
  begin

    -- Senyales por defecto
    WrPaleta <= '1';
    incrementaIndiceInit <= '0';
    progDatosPaleta <= (others => '0');
    progRS <= "001";
    latchDatosPaleta <= '1';
    latchRS <= '1';
    finProgramacion <= '0';

    case estadoActual is
      when stInicial =>
        -- Espera hasta que se le de la senyal de inicioProgramacion desde otro proceso
        if (inicioProgramacion = '1') then
          estadoSiguiente <= stEscribe;
          progRS <= initRS(indiceInit);
          progDatosPaleta <= initDAC(indiceInit);
        else
          estadoSiguiente <= stInicial;
          latchDatosPaleta <= '0';
          latchRS <='0';
      end if;

      when stEscribe =>
        -- Mantenemos las senyales de seleccion y de datosPaleta para el ciclo de escritura
        estadoSiguiente <= stCicloEscritura;
        progRS <= initRS(indiceInit);
        progDatosPaleta <= initDAC(indiceInit);
        WrPaleta <= '0';

      when stCicloEscritura =>
        -- Si ya hemos finProgramacion la inicializacion, hemos finProgramacion de configurarlo
        --  si no, hacemos un nuevo ciclo
        if (indiceInit = 5) then
          estadoSiguiente <= stInicial;
          finProgramacion <= '1';
        else
          estadoSiguiente <= stSiguienteEscribe;
        end if;

        --Mantenemos las senyales para asegurar que los tiempos de hold no se violan y
pasamos al siguiente indice
        progRS <= initRS(indiceInit);
        progDatosPaleta <= initDAC(indiceInit);
        incrementaIndiceInit <= '1';

      when stSiguienteEscribe =>
        estadoSiguiente <= stEscribe;
        progRS <= initRS(indiceInit);
        progDatosPaleta <= initDAC(indiceInit);

    end case;

  end process;
-------------------------------------


-------------------------------------
-- Asignamos las lineas cuando necesiten ser latcheadas.
-- En otro caso, las mantenemos en alta impedancia para crear un buffer triestado
-------------------------------------
datosPaleta <= progdatosPaleta when latchDatosPaleta = '1'
      else (others => 'Z');
RS <= progRS when latchRS = '1'
      else (others => 'Z');
-------------------------------------


end programaRamDacTrueColor_Arquitectura;
```

registroGenerico.vhd

```
--------------------------------------------------------------------------------
-- REGISTRO GENÉRICO DE N BITS, SINCRONO, RESET ASINCRONO Y SENYAL DE
CARGA
--------------------------------------------------------------------------------
-- ENTIDAD: RegistroGenerico
-- CONEXIONES:
-- PUERTOS ENTRADA:
--    clk: Entrada del reloj a 50MHz (activo con falnco de subida)
--    rst: Reset del circuito (activo a baja)
--    cargar: Senyal que permite la carga del registro (activa a alta)
--    entrada(n-1..0): Líneas de entrada al registro
-- PUERTOS SALIDA:
--    salida(n-1..0): Líneas de salida al registro
--------------------------------------------------------------------------------
-- Este modulo implementa un registro sincrono por flanco de subida estandar.
-- El reset es asincrono y activo a alta y coloca la salida a cero.
-- Tiene tambien la correspondiente senyal de carga (activa a alta) que permite
--  que se carge en el registro lo que haya en sus lineas de entrada o no.
--------------------------------------------------------------------------------

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


--------------------------------------------------------------------------------
-- Declaracion de la entidad
--------------------------------------------------------------------------------
entity RegistroGenerico is
  generic (n: integer:=16);
  port(
    clk: in std_logic;     -- Reloj del registro (activo en flancos de subida)
    rst: in std_logic;     -- Reset del circuito (activo a baja)
    cargar:in std_logic;   -- Senyal de capacitacion de la carga
    entrada: in std_logic_vector(n-1 downto 0);
    salida : out std_logic_vector(n-1 downto 0)
  );
end RegistroGenerico;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- Declaracion de la arquitectura
--------------------------------------------------------------------------------
architecture RegistroGenerico_Aquitectura of RegistroGenerico is

begin

elRegistroGenerico:
process (clk,rst)
  begin
    if (rst = '0') then
      salida <= (others => '0');
    elsif (clk'event and clk='1' ) then
      -- Solo responde en flancos de subida, cuando este capacitado
   if (cargar = '1') then
        salida <= entrada;
      end if;
    end if;
end process;

end RegistroGenerico_Aquitectura;
```

video.ucf

```
#########################################################
###  CONEXIONES DE LOS PUERTOS DEL MODULO GENERAL  ###
#########################################################
# - En este fichero tenemos todas las conexiones posibles que podemos hacer con
#    la placa por lo que unicamente tendremos que comentar/descomentar las que
#    nos convenga
#########################################################


###############
###  RELOJ  ###
###############
NET relojGeneral LOC = P89;


##############################
###  BANCO IZQUIERDO SRAM  ###
##############################
NET CSI LOC = P186;                # Senyales de control
NET OEI LOC = P228;
NET WEI LOC = P201;
NET datosMemoriaI<0> LOC = P202;     # datosMemoria[15..0]
NET datosMemoriaI<1> LOC = P203;
NET datosMemoriaI<2> LOC = P205;
NET datosMemoriaI<3> LOC = P206;
NET datosMemoriaI<4> LOC = P207;
NET datosMemoriaI<5> LOC = P208;
NET datosMemoriaI<6> LOC = P209;
NET datosMemoriaI<7> LOC = P215;
NET datosMemoriaI<8> LOC = P216;
NET datosMemoriaI<9> LOC = P217;
NET datosMemoriaI<10> LOC = P218;
NET datosMemoriaI<11> LOC = P220;
NET datosMemoriaI<12> LOC = P221;
NET datosMemoriaI<13> LOC = P222;
NET datosMemoriaI<14> LOC = P223;
NET datosMemoriaI<15> LOC = P224;
NET direccionMemoriaI<0> LOC = P200;  #direccionMemoriaI[18..0]
NET direccionMemoriaI<1> LOC = P199;
NET direccionMemoriaI<2> LOC = P195;
NET direccionMemoriaI<3> LOC = P194;
NET direccionMemoriaI<4> LOC = P193;
NET direccionMemoriaI<5> LOC = P192;
NET direccionMemoriaI<6> LOC = P191;
NET direccionMemoriaI<7> LOC = P189;
NET direccionMemoriaI<8> LOC = P188;
NET direccionMemoriaI<9> LOC = P187;
NET direccionMemoriaI<10> LOC = P238;
NET direccionMemoriaI<11> LOC = P237;
NET direccionMemoriaI<12> LOC = P236;
NET direccionMemoriaI<13> LOC = P235;
NET direccionMemoriaI<14> LOC = P234;
NET direccionMemoriaI<15> LOC = P232;
NET direccionMemoriaI<16> LOC = P231;
NET direccionMemoriaI<17> LOC = P230;
NET direccionMemoriaI<18> LOC = P229;

##############################
###  BANCO DERECHO SRAM  ###
##############################
NET CSD LOC = P109;                # Senyales de control
NET OED LOC = P95;
NET WED LOC = P68;
NET datosMemoriaD<0> LOC = P70;          # datosMemoriaD[15..0]
NET datosMemoriaD<1> LOC = P71;
NET datosMemoriaD<2> LOC = P72;
NET datosMemoriaD<3> LOC = P73;
NET datosMemoriaD<4> LOC = P74;
NET datosMemoriaD<5> LOC = P78;
NET datosMemoriaD<6> LOC = P79;
NET datosMemoriaD<7> LOC = P80;
NET datosMemoriaD<8> LOC = P81;
NET datosMemoriaD<9> LOC = P82;
NET datosMemoriaD<10> LOC = P84;
NET datosMemoriaD<11> LOC = P85;
NET datosMemoriaD<12> LOC = P86;
NET datosMemoriaD<13> LOC = P87;
NET datosMemoriaD<14> LOC = P93;
NET datosMemoriaD<15> LOC = P94;
```

```
NET direccionMemoriaD<0> LOC = P67;       # direccionMemoriaD[18..0]
NET direccionMemoriaD<1> LOC = P66;
NET direccionMemoriaD<2> LOC = P65;
NET direccionMemoriaD<3> LOC = P64;
NET direccionMemoriaD<4> LOC = P63;
NET direccionMemoriaD<5> LOC = P57;
NET direccionMemoriaD<6> LOC = P56;
NET direccionMemoriaD<7> LOC = P55;
NET direccionMemoriaD<8> LOC = P54;
NET direccionMemoriaD<9> LOC = P53;
NET direccionMemoriaD<10> LOC = P108;
NET direccionMemoriaD<11> LOC = P107;
NET direccionMemoriaD<12> LOC = P103;
NET direccionMemoriaD<13> LOC = P102;
NET direccionMemoriaD<14> LOC = P101;
NET direccionMemoriaD<15> LOC = P100;
NET direccionMemoriaD<16> LOC = P99;
NET direccionMemoriaD<17> LOC = P97;
NET direccionMemoriaD<18> LOC = P96;

########################
###  VIDEO  DECODER ###
########################
#NET llc LOC = P92;
#NET rts<0> LOC = P111;
#NET rts<1> LOC = P110;
#NET rtco LOC = P113;
#NET vpo<0> LOC = P116;
#NET vpo<1> LOC = P117;
#NET vpo<2> LOC = P118;
#NET vpo<3> LOC = P125;
#NET vpo<4> LOC = P126;
#NET vpo<5> LOC = P127;
#NET vpo<6> LOC = P128;
#NET vpo<7> LOC = P130;
#NET scl LOC = P114;
#NET sda LOC = P115;


################
###  RAMDAC  ###
################
NET pixelClock LOC = P52;    # Reloj
NET hSync LOC = P48;         # Senyales de sincronizacion
NET vSync LOC = P49;
NET blanking LOC = P50;
```

```
#NET pixel<0> LOC = P70;     # Datos a mostrar (compartido con banco derecho)
#NET pixel<1> LOC = P71;
#NET pixel<2> LOC = P72;
#NET pixel<3> LOC = P73;
#NET pixel<4> LOC = P74;
#NET pixel<5> LOC = P78;
#NET pixel<6> LOC = P79;
#NET pixel<7> LOC = P80;
#PARA CONFIGURAR EL RAMDAC
NET RdPaleta LOC = P47;      # Senyales de control
NET WrPaleta LOC = P46;
NET RS<0> LOC = P31;         # RS[2..0]
NET RS<1> LOC = P28;
NET RS<2> LOC = P26;
NET datosPaleta<0> LOC = P42;  # datosPaleta[7..0]
NET datosPaleta<1> LOC = P41;
NET datosPaleta<2> LOC = P40;
NET datosPaleta<3> LOC = P39;
NET datosPaleta<4> LOC = P38;
NET datosPaleta<5> LOC = P36;
NET datosPaleta<6> LOC = P35;
NET datosPaleta<7> LOC = P34;


################
###  STEREO  ###
################
#NET mclk LOC = P3;
#NET lrck LOC = P4;
#NET sclk LOC = P5;
#NET sdin LOC = P6;
#NET sdout LOC = P7;


##################
###  ETHERNET  ###
##################
#NET col LOC = P23;
#NET crs LOC = P21;
NET ethernet LOC = P24;    # La desactivaremos para evitar contencion
#NET tx_clk LOC = P210;
#NET tx_en LOC = P25;
#NET tx_er LOC = P27;
```

```
#NET txdata<0> LOC = P42;
#NET txdata<1> LOC = P41;
#NET txdata<2> LOC = P40;
#NET txdata<3> LOC = P39;
#NET txdata<4> LOC = P31;
#NET rx_clk LOC = P213;
#NET rx_dv LOC = P26;
#NET rx_er LOC = P28;
#NET rxdata<0> LOC = P38;
#NET rxdata<1> LOC = P36;
#NET rxdata<2> LOC = P35;
#NET rxdata<3> LOC = P34;
#NET rxdata<4> LOC = P33;
#NET fds/mdint LOC = P18;
#NET mdc LOC = P19;
#NET mdio LOC = P20;


####################
###  PULSADORES  ###
####################
NET resetGeneral LOC = P174;
#NET efecto LOC = P175;
NET pausa LOC = P176;
#NET pb4 LOC = P185;


##################
###  SWITCHES  ###
##################
#NET dip1 LOC = P161;
#NET dip2 LOC = P159;
#NET dip3 LOC = P155;
#NET dip4 LOC = P153;
#NET dip5 LOC = P149;
#NET dip6 LOC = P146;
#NET dip7 LOC = P142;
#NET dip8 LOC = P140;
```

```
##############################
###  DISPLAYS 7-SEGMENTOS  ###
##############################
#NET lhex<0> LOC = P177;   # izquierdo
#NET lhex<1> LOC = P167;
#NET lhex<2> LOC = P163;
#NET lhex<3> LOC = P156;
#NET lhex<4> LOC = P145;
#NET lhex<5> LOC = P138;
#NET lhex<6> LOC = P134;
#NET rhex<0> LOC = P124;   # derecho
#NET rhex<1> LOC = P132;
#NET rhex<2> LOC = P133;
#NET rhex<3> LOC = P139;
#NET rhex<4> LOC = P141;
#NET rhex<5> LOC = P144;
#NET rhex<6> LOC = P147;


##############
###  LEDS  ###
##############
#NET bar0 LOC = P152;
#NET bar1 LOC = P154;
#NET liminf LOC = P157;
NET estadofsms<0> LOC = P160;
NET estadofsms<1> LOC = P162;
NET estadofsms<2> LOC = P169;
NET estadofsms<3> LOC = P168;
NET estadofsms<4> LOC = P173;
#NET limsup LOC = P131;
#NET bar9 LOC = P171;


######################
###  TECLADO PS/2  ###
######################
NET relojTeclado LOC = P13;
NET datosTeclado LOC = P17;
```

```
#############
###  USB  ###
#############
#NET usb_oen LOC = P12;
#NET usb_vpo LOC = P13;
#NET usb_vmo/fseo LOC = P17;
#NET usb_rcv LOC = P11;
#NET usb_vp LOC = P10;
#NET usb_vm LOC = P9;
#NET usb_suspnd LOC = P12;


#########################
###  PUERTO PARALELO  ###
#########################
#NET ppdata<0> LOC = P177;
#NET ppdata<1> LOC = P167;
#NET ppdata<2> LOC = P163;
#NET ppdata<3> LOC = P156;
#NET ppdata<4> LOC = P145;
#NET ppdata<5> LOC = P138;
#NET ppdata<6> LOC = P134;
#NET ppdata<7> LOC = P124;
#NET ppstatus<3> LOC = P132;
#NET ppstatus<4> LOC = P133;
#NET ppstatus<5> LOC = P139;
#NET ppstatus<6> LOC = P141;


##################
###  FLASH RAM ###
##################
#NET fcen LOC = P170;       # Senyales de control
#NET foen LOC = P173;
#NET fwen LOC = P131;
#NET frdy LOC = P171;
#NET fdata<0> LOC = P177;   # datos[7..0]
#NET fdata<1> LOC = P167;
#NET fdata<2> LOC = P163;
#NET fdata<3> LOC = P156;
#NET fdata<4> LOC = P145;
#NET fdata<5> LOC = P138;
#NET fdata<6> LOC = P134;
#NET fdata<7> LOC = P124;
```

```
#NET faddr<0> LOC = P132;   # direccion[20..0]
#NET faddr<1> LOC = P133;
#NET faddr<2> LOC = P139;
#NET faddr<3> LOC = P141;
#NET faddr<4> LOC = P144;
#NET faddr<5> LOC = P147;
#NET faddr<6> LOC = P152;
#NET faddr<7> LOC = P154;
#NET faddr<8> LOC = P157;
#NET faddr<9> LOC = P160;
#NET faddr<10> LOC = P162;
#NET faddr<11> LOC = P169;
#NET faddr<12> LOC = P168;
#NET faddr<13> LOC = P161;
#NET faddr<14> LOC = P159;
#NET faddr<15> LOC = P155;
#NET faddr<16> LOC = P153;
#NET faddr<17> LOC = P149;
#NET faddr<18> LOC = P146;
#NET faddr<19> LOC = P142;
#NET faddr<20> LOC = P140;
```

Bibliografía

— Captura y Codificación de Video para PC
Universidad Politécnica de Cataluña, 2001

— Digital Video and Image Processing
http://www.xilinx.com/esp/prof_brdcst/collateral/videoprocessing.pdf

— Diseño Automático de Sistemas. Documentación
Facultad de Informática. Universidad Complutense de Madrid
http://dacya.ucm.es/mendias/DAS

— DSP Central
http://university.xilinx.com/dsp

— Hojas de Especificaciones de la Placa Virtex XSV-800 y sus Componentes
http://www.xess.com/FPGA/homepage.html

— **Página Principal de Xilinx**
http://www.xilinx.com

— Prototipado de Sistemas Digitales en Hardware Reconfigurable
Gómez Pulido, J. A.
Universidad de Extremadura, 2000

— Rapid Prototyping of Digital Systems
M. Kropidlowski, A. Handkiewicz, M.Lukowiak
Institute of Control and System Engineering, Poznan. Polonia

— VHDL Interfaces and Example Designs for the XSV Board 1.0
School of Computer Science and Electrical Engineering,
University of Queensland, Brisbane. Australia
http://www.csee.uq.edu.au

— XESS Example Desingn. Tutorials and Application Notes
http://www.xess.com

— Xilinx Emerging Standards and Protocols
http://university.xilinx.com/esp

Para contactar con los autores:

Rodrigo Borrego Alonso	rodrigo@cielda.com
Sara Hernández García	sara_hernandez@terra.es
José Luis Moisés González	jlmoigon@hotmail.com
Carolina Valdazo Ampuero	carol_valdazo@hotmail.com