Universidad Complutense de Madrid

Facultad de Ciencias Físicas

Ingeniería Electrónica

Proyectos

2002

Desarrollo de soft IP cores para el control de dispositivos de E/S

Francisco Javier Pérez Gómez

Tutores:

Jose Manuel Mendías Cuadros

Hortensia Mecha López

INDICE

1. INTRODUCCIÓN.

Es bien sabido que la tendencia a implementar plataformas informáticas para control de procesos es generalizada.

El campo del video y tratamiento de imágenes se incluye dentro de esta disciplina. Tradicionalmente las señales de video y las utilizadas en los ordenadores no eran compatibles, se hacen necesarios elementos que sean capaces de procesar las señales para su posterior tratamiento a través de un microprocesador.

El objetivo de este proyecto persigue la implementación de un core hardware codificado en VHDL y proyectado sobre una FPGA capaz de transformar una señal de video estándar en un formato adecuado para su posterior tratamiento y visualización en un monitor VGA. Para ello dispondremos de herramientas hardware y software que permitan la construcción de un módulo que permita dicha conversión.

La creación de un IP (Intelectual Property) core permite su reutilización en otros proyectos, pudiendo formar parte integrante de ellos como componente.

Se desarrollan una serie de módulos que permiten la decodificación de señales de video digital, la programación de dispositivos a través de I2c, así como la generación de señales de video progresivo compatibles con cualquier dispositivo VGA (Video Graphics Adapter).

En concreto disponemos de una placa de prototipado que integra un dispositivo de lógica programable, además de otros circuitos integrados específicos. El dispositivo de lógica programable (FPGA), necesita una configuración para realizar la función deseada, dicho configuración podrá ser generado y depurado por medio de un entorno de desarrollo adecuado, el lenguaje de diseño usado será VHDL.

La memoria se divide en una serie de apartados tratando los siguientes puntos:

2. Fundamentos, se exponen las características de las señales de vídeo y de algunos de los dispositivos necesarios para su manipulación y tratamiento.
3. Las herramientas y el entorno de desarrollo utilizado para su realización.
4. Diseño, dónde se comentan las consideraciones necesarias y se detallan cada uno de los módulos que lo componen.
5. Las conclusiones y posibles ampliaciones a realizar.
6. Referencias, bibliografía y recomendaciones de organismos reguladores dónde se especifican las características de una señal de vídeo para su difusión.
7. Un apéndice que recoge todos los ficheros de diseño.

2. FUNDAMENTOS.

2.1 Señales de video.

Históricamente, la evolución de las señales de video han estado condicionadas por requerimientos de ancho de banda. Tanto en la transmisión de una señal de video como en su almacenamiento era prioritario reducir los costes necesarios.

Una señal de video se compone de 3 colores primarios Rojo, Verde y Azul, la combinación de estos tres colores permite obtener cualquier color del espectro desde el negro al blanco. Cuando una señal se compone de los tres colores mencionados se denomina señal en componentes RGB, esta señal necesita para su transmisión de 3 hilos conductores (uno para componente de color), se hace también necesario que los tres canales de transmisión tengan el mismo ancho de banda por que las tres componentes son tratadas por igual.

Las señales de video aparte de tener información de imagen tienen una información de temporización para permitir la correcta reproducción sobre un monitor, dichas referencias de temporización son los llamados sincronismos horizontales y verticales. Un monitor necesita estas señales ya que representa una imagen mediante un barrido de las sucesivas líneas que conforman una imagen desde la primera a la última para volver a empezar desde el principio. Son estas señales llamadas sincronismos las que permiten que la imagen se centre correctamente sobre el monitor. Durante estas referencias de temporización el haz de electrones necesita un cierto tiempo para volver a su posición original y así

comenzar una nueva línea o una nueva imagen. A este tiempo se denomina retrazado y la señal de video permanece borrada. En la siguiente figura se aprecia una imagen de video con la zona de imagen activa y los borrados tanto horizontales como verticales.

Se observa que antes y después de cada pulso de sincronismo tanto horizontal como vertical existe una zona de borrado, la zona de borrado cubre un tiempo anterior y posterior al pulso de sincronismo.

Estudios determinan que el ojo humano es mas sensible a variaciones de luminosidad que a variaciones de color, se puede aprovechar este efecto si transformamos la señal de video en una componente que defina la luminosidad de la señal y otras que introduzcan color a dicha señal. Se crean diferentes espacios de color.

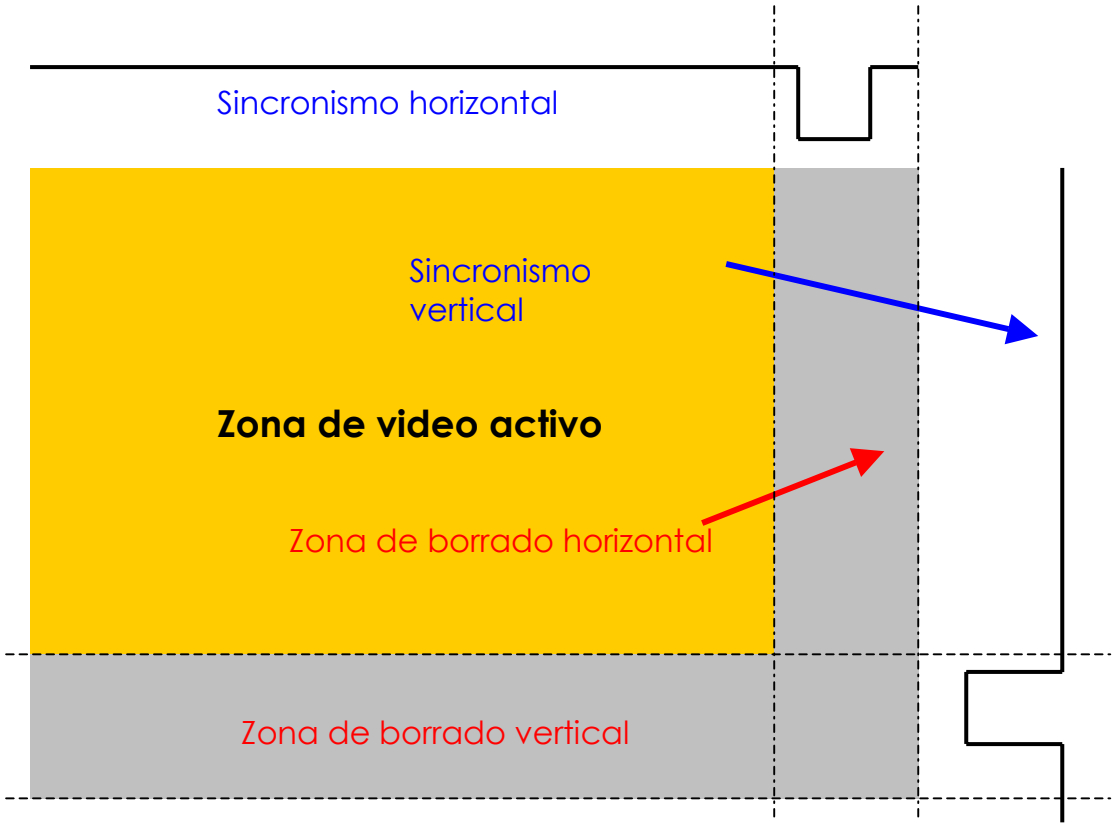


Figura 1. Representación de una imagen y sus sincronismos.

Los espacios de color mas utilizados son el (RGB) y (Y,B-Y,R-Y). Los dos espacios de color se relacionan mediante unas ecuaciones que ponderan cada una de las componentes de un espacio de color para obtener el otro, la ventaja de utilizar (Y,B-Y,R-Y) radica en que la componente Y por sí sola es capaz de representar una imagen en escala de grises, las otras dos componentes B-Y y R-Y añaden la información de color para completar la imagen, este tipo de espacios de color permiten compatibilizar sistemas en color con sistemas en blanco y negro.

En la siguiente figura se puede ver la relación existente entre estos dos espacios de color, el paso de un espacio a otro se denomina comúnmente matrizado.

Debido a que el ojo humano es menos sensible a variaciones de color que de luminosidad se puede reducir el ancho de banda de las señales que llevan información de color a la mitad, con lo que se consigue una representación de la imagen con una diferencia de calidad inapreciable respecto a RGB reduciendo los recursos necesarios para su transmisión y almacenamiento.

Este formato de la señal de video, en cualquier espacio de color se denomina “video en componentes” ya que se utilizan varios canales para transmitir la señal de video.

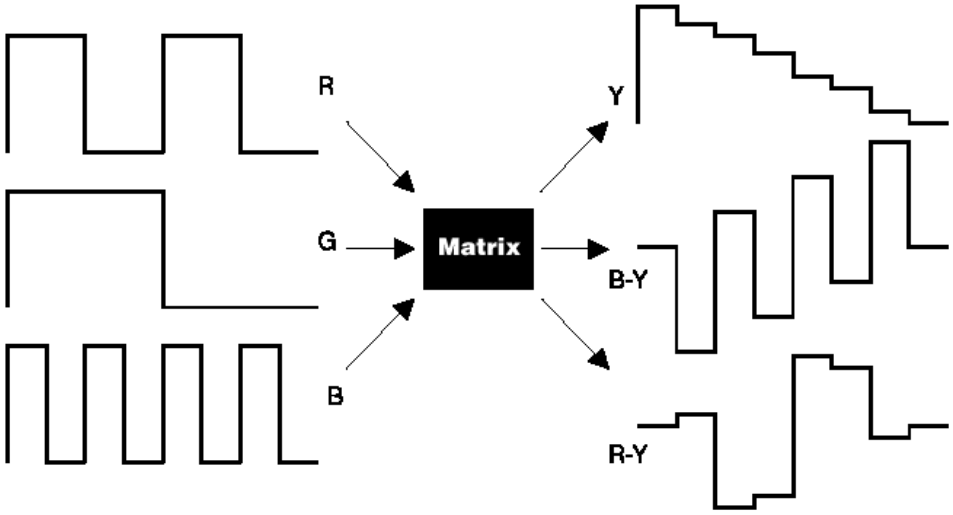


Figura 2. Matrizado de una señal RGB en Y, B-Y, R-Y.

Un logro más a la hora de reducir el ancho de banda ocupado por una señal de video es la utilización de “video compuesto”. Una señal de video compuesto consiste, básicamente, en modular las dos componentes de color con una portadora en cuadratura para cada componente y montar la señal resultante sobre la señal de luminancia. En el extremo receptor se hará necesario disponer de un circuito para demodular la señal en sus componentes para así poder visualizarla. En última instancia un tubo de rayo catódicos (CRT) tanto de un televisor como de un PC trabajan en RGB, que es la representación nativa de una señal de video.

Indicar que la captación de una señal de video por medio de una cámara o cualquier otro dispositivo ó generación sintética de la señal en PC se realiza en RGB y su visualización en un CRT también se produce en RGB, todas las transformaciones intermedias que se realicen sobre la señal de video deberán ser invertidas por el extremo receptor.

Modular las componentes de color sobre la luminancia (Y) permite la utilización de un solo conductor para transmitir la señal de video y reduce los requisitos de ancho de banda de transmisión y espacio requerido para su almacenamiento.

En la siguiente figura se puede apreciar un diagrama de bloques básico de la constitución de un modulador de video compuesto.

La notación (Y, R-Y, B-Y) suele ser equivalente, salvo pequeños valores de desplazamiento a (YUV) para designar espacios de color, nosotros las consideraremos equivalentes.

Existe básicamente 3 sistemas de modulación de video PAL, NTSC y SECAM, en la figura se representa un modulador PAL.

Para aclarar conceptos veamos cual es el proceso de transformación de una señal RGB a su equivalente en video compuesto utilizando una modulación y formato de 625 líneas, el formato utilizado en Europa. Concretamente codificaremos una señal de barras de color.

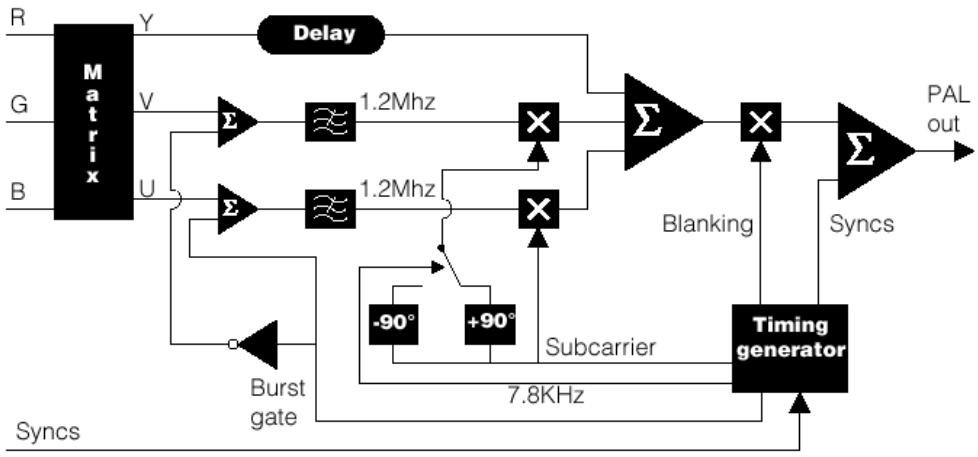


Figura 3. Arquitectura básica de un modulador de video compuesto.

Figura 4. Señal de barras de color en pantalla.

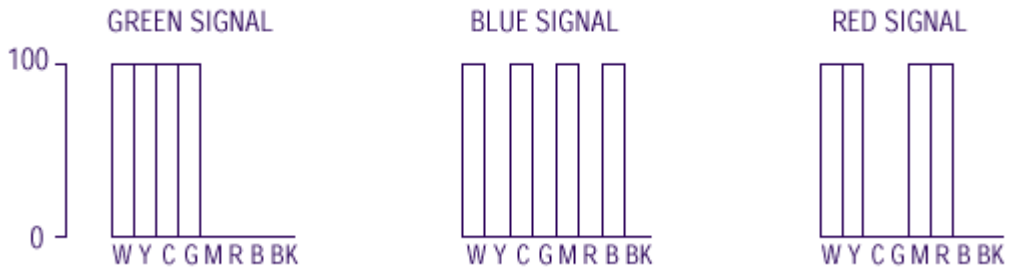


Figura 5. Señales RGB correspondientes a una carta de barras

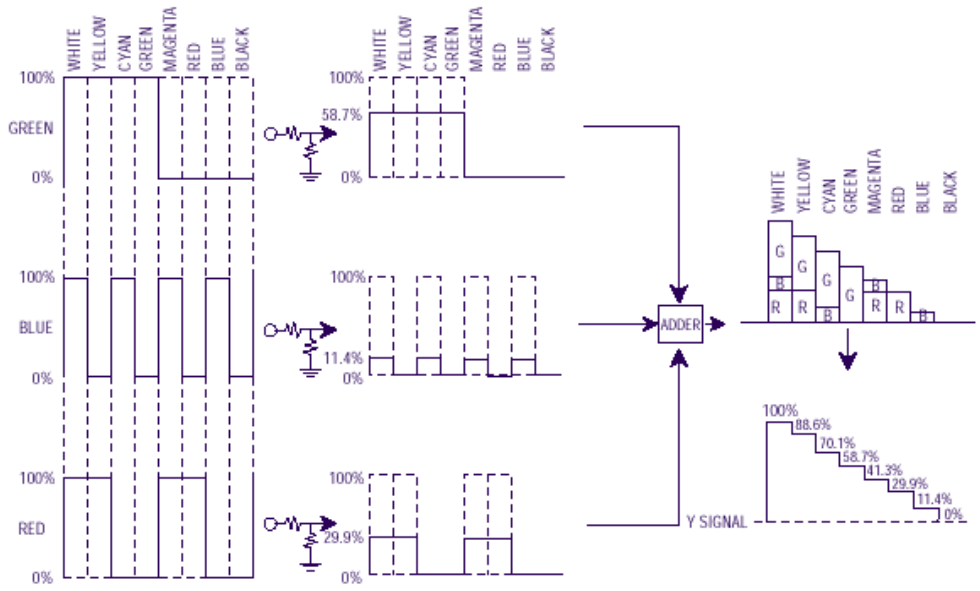


Figura 6. Formación de la señal de luminancia Y.

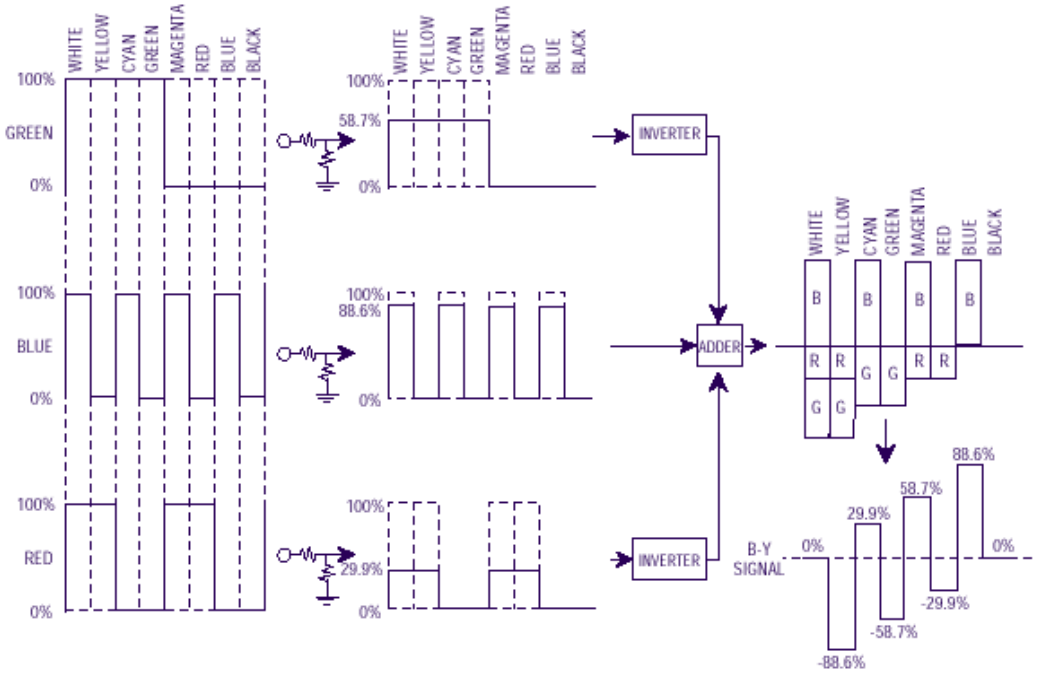


Figura 7. Formación de la componente B-Y.

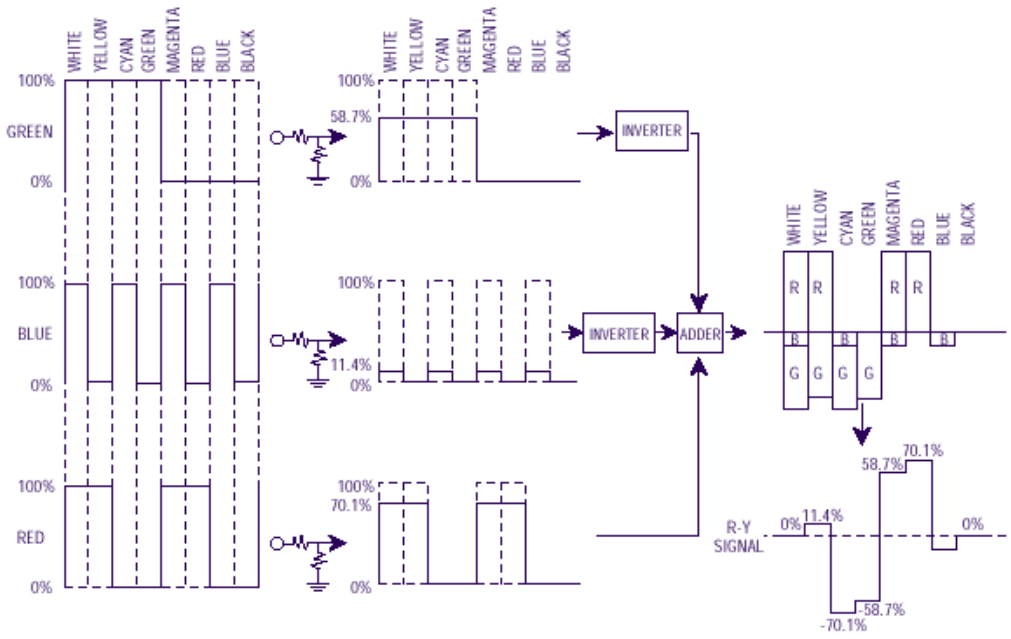


Figura 8. Formación de la componente R-Y.

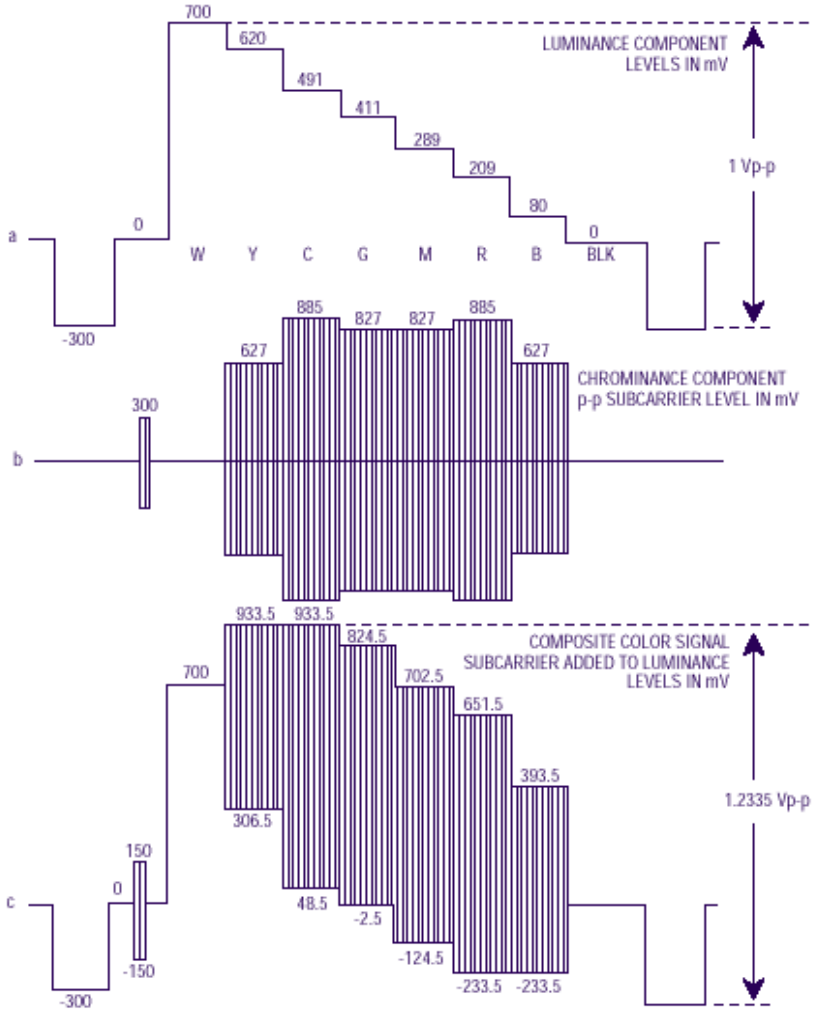


Figura 9. Señal de video compuesto obtenida.

En las anteriores figura se aprecian los pasos necesarios para obtener una señal de video compuesto. Partiendo de las señales en componentes RGB se realiza una transformación de espacio de color a Y, B-Y, R-Y a través de un circuito de matrizado. Las señales de color son moduladas con una portadora y añadidas a la señal de luminancia para formar la señal final. En la figura 8 se representa una línea de video, se aprecia también que se incluye un pulso al inicio de la misma llamado sincronismo horizontal que permite el barrido de la pantalla.

Las anteriores transformaciones de la señal de video reducen las necesidades de ancho de banda necesaria para su transmisión aprovechando ciertas características del ojo humano y propiedades de modulación. Existen formas adicionales de reducir dicho ancho de banda aprovechando ciertas propiedades temporales de la señal de video.

Una señal de video es representada en un CRT mediante barridos periódicos del haz de electrones del monitor. La frecuencia de refresco del monitor es determinante para reducir efectos de parpadeo en la pantalla. Cuanto mayor sea la frecuencia de refresco del monitor menos apreciables serán los efectos debidos al parpadeo, sin embargo, cuanto mayor sea esta frecuencia de refresco mas ancho de banda de transmisión necesitará nuestra señal. Una solución utilizada por los sistemas de video se denomina barrido entrelazado, el barrido entrelazado consiste en dividir una imagen en líneas pares e impares y presentarlas sobre la pantalla alternativamente.

Típicamente, una señal de video consta de 525 ó 625 líneas, dependiendo del formato utilizado. Al número total de líneas que forman una imagen se llama “cuadro”, si dividimos un cuadro en sus líneas pares e impares obtenemos 2 “campos” llamados campo par y campo impar respectivamente. De esta manera, se puede doblar la frecuencia de refresco del monitor basándonos en una representación por campos en lugar de por cuadros.

En la siguiente figura se aprecia el barrido entrelazado para un sistema de 525 lineas.

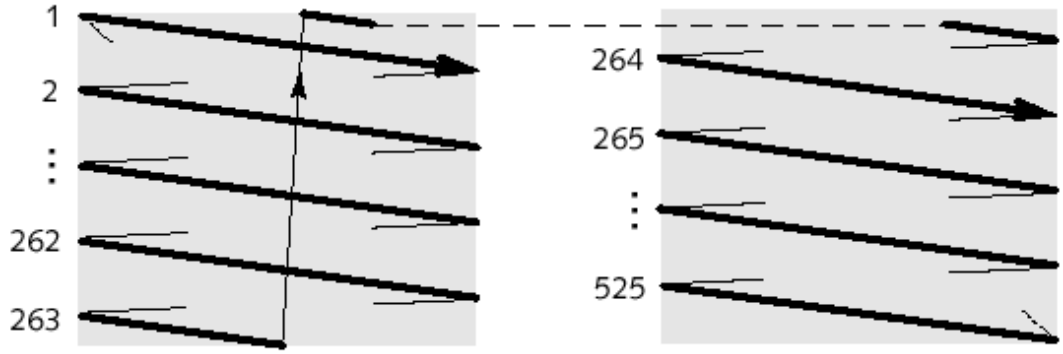


Figura 10. Estructura de un barrido entrelazado.

Como se aprecia en la figura, la representación de una imagen completa sucede en dos fases, visualización de campo impar y campo par, se mejora el efecto parpadeo y se logra mantener el ancho de banda de la señal de video. Como veremos mas adelante, el formato de video en las computadoras mantiene un barrido progresivo en el que se presenta la imagen completa en cada barrido del monitor, son necesarias altas frecuencias de refresco para evitar el efecto parpadeo con el consiguiente aumento del ancho de banda necesario para transmisión.

Para ver las diferencia entre los barridos, se presentan a continuación una imagen con barrido entrelazado y con barrido progresivo.

El barrido progresivo describe una imagen con una pasada secuencial de todas las líneas.

Las señales de video que se manejan en los entornos de computadoras suelen ser en componentes, con espacio de color RGB, altas resoluciones, barridos progresivos y altas frecuencias de refresco, para poder visualizar una señal de video en un monitor digital son necesarias todas estas adaptaciones, algunas de las cuales serán objeto de realización en este proyecto.

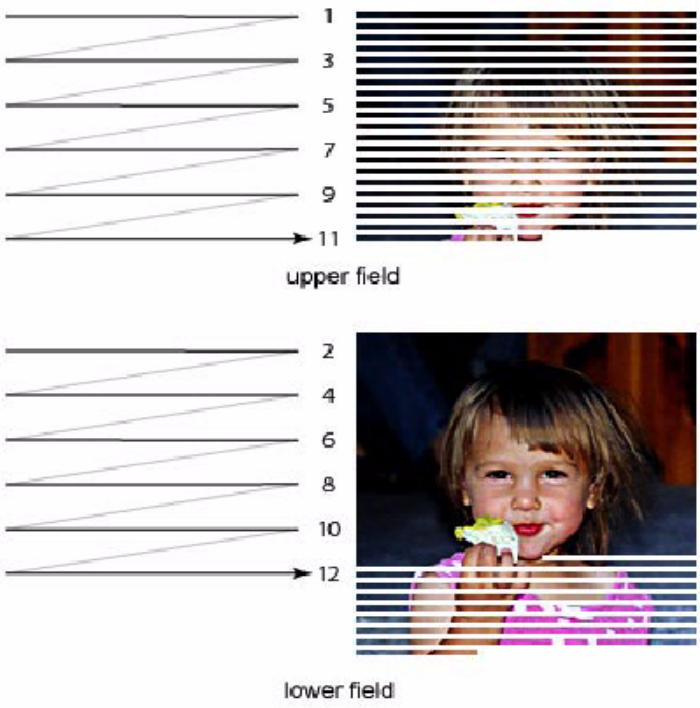


Figura 11. Barrido entrelazado.

El barrido entrelazado describe una imagen completa con dos pasadas con líneas alternadas.

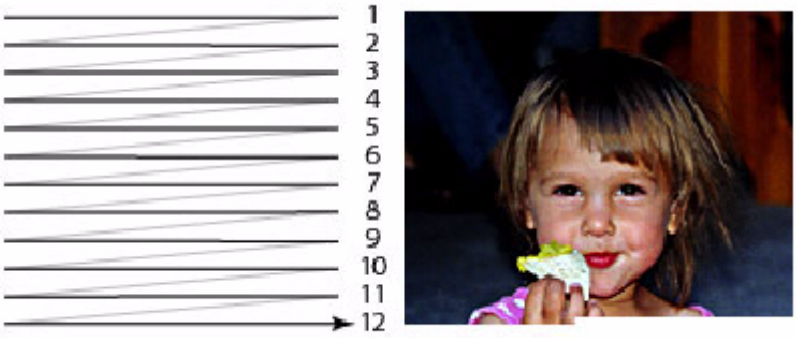


Figura 12. Barrido progresivo.

En concreto para poder visualizar una señal de video en un monitor VGA necesitamos demodular la señal de video compuesto, convertirla a un espacio de color RGB y transformarla de barrido entrelazado a progresivo. Para la realización de dichas operaciones nos apoyaremos en hardware dedicado a estas funciones y un algoritmo de control que se implementará en la FPGA.

A continuación se presentan algunas nociones sobre los elementos utilizados.

2.2 Señal video digital ITU-R BT.656

La señal de video una vez digitalizada, demodulada y separadas sus componentes es formateada para construir una trama que permita su reconstrucción en cualquier elemento receptor compatible. La trama es formada por palabras de 8 bits y están multiplexadas en el tiempo las muestras correspondientes a luminancia y las dos componentes de color B-Y, R-Y. Como hemos visto una señal de video tiene además de información sobre la señal de video a visualizar (llamada línea activa) otra serie de informaciones como sincronismos horizontales y verticales y sus zonas de borrado. En esas zonas de borrado no puede incluirse información sobre señal de video. Existe una correspondencia entre la línea de video analógica y la digital.

En la trama digital se incluyen las referencias de sincronización correspondientes a una señal analógica por medio de la utilización de palabras reservadas únicamente a este fin, estos valores no pueden ser usados en la zona activa de video. Estas palabras se denominan SAV (Start of Active Video) para indicar que a partir de ese momento comienzan muestras que corresponden a señal de video a visualizar y EAV (End of Active Video) para indicar que entramos en zona de borrado.

Las palabras reservadas siguen una secuencia de 4 muestras denominada TRS (Timming Reference Signal) y que nos sirven para identificar las zonas de imagen que estamos tratando, sincro horizontal o vertical, zona de video activo, borrado o comienzo de un nuevo frame.

Una descripción más detallada de la secuencia digital respecto a la analógica se representa en la siguiente figura donde se puede apreciar el TRS como la secuencia de 4 muestras con valores 3FF,000,000,XYZ dónde cada palabra se codifica en 10 bits (nosotros usaremos los 8 bits más significativos) y dónde XYZ contiene información relevante a la situación que nos encontramos dentro de un cuadro de vídeo.

Se puede apreciar también como las palabras que no forman parte de TRS transportan la información de video en forma multiplexada en el tiempo en forma de muestras de Y, Cr y Cb para la luminancia y cada componente de color respectivamente.

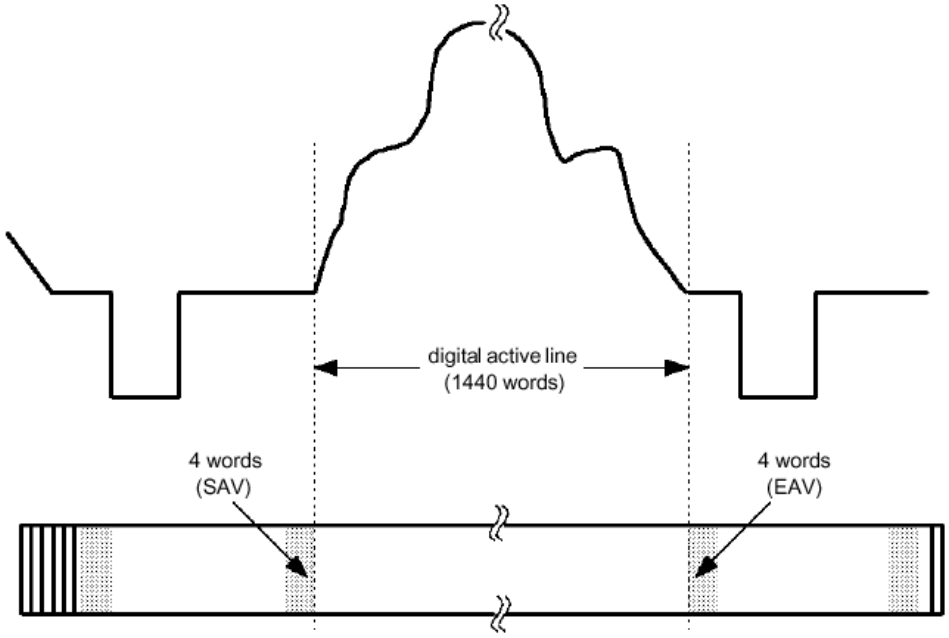


Figura 13. Correspondencia de línea analógica y digital.

Debido a la reducción de ancho de banda de las componentes de color existen mas muestras de luminancia que de cada componente de color, se observa que la secuencia de muestras es Cb,Y,Cr,Y,Cb,y,Cr,Y... A esta estructura de muestreo se denomina 4:2:2 indicando que cada 4 muestras de luminancia existen 2 de cada componente de color, a la hora de convertir a RGB las muestras que faltan de cada componente de color serán obtenidas por interpolación.

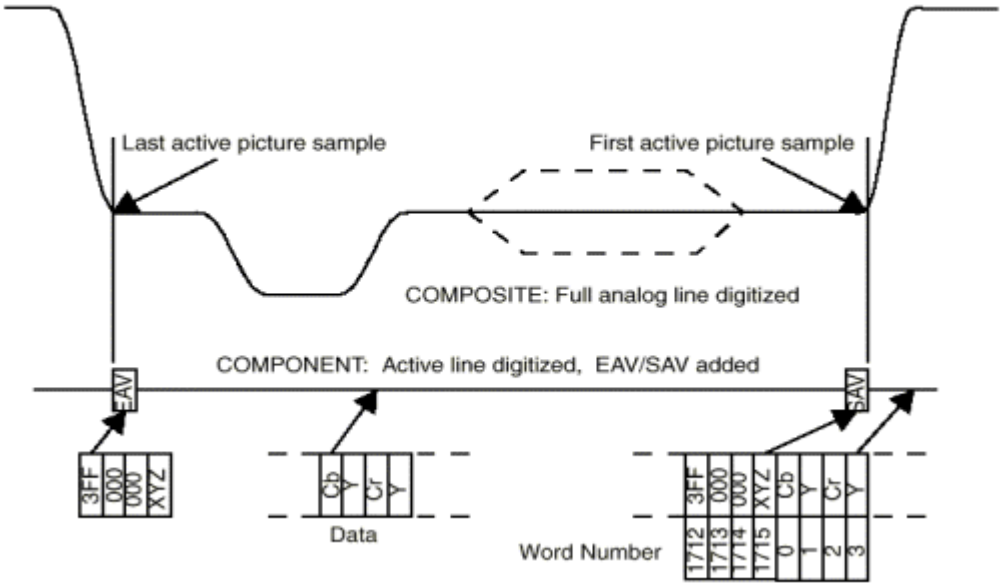


Figura 14. Descripción detallada de TRS y multiplexado de componentes.

Figura 15. Estructura de muestreo 4:2:2.

2.3 Decodificador video compuesto.

Un decodificador de video compuesto tiene como entrada una señal de video como las comentadas anteriormente y realiza las siguientes operaciones:

- Separación de componentes de luminancia (Y) y componentes de color (U,V).
- Demodulación de las componentes de color y descomposición en U y V.

Dependiendo de las características del decoder utilizado, puede entregar a su salida señales en componentes analógicas en espacio de color YUV o, contando con un circuito de matrizado interno generar las correspondientes RGB.

Tambien existen los llamados decoder digitales, que realizan todas estas operaciones en el dominio digital. La señal de entrada en video compuesto es muestreada y cuantizada, unos sintetizadores digitales de frecuencia generan las correspondientes frecuencias de portadora y realizan una demodulación digital. Entregan a su salida un tren de muestras digitales correspondientes a las componentes de la señal de video compuesto de entrada. Estas muestras digitales pueden corresponder a un espacio de color YUV o RGB.

Un diagrama de bloques de un decoder como el utilizado en el proyecto se representa en la figura 12. Consta de un bloque que separa la señal de luminancia y componentes de color, las componentes de color o crominancia sufren un proceso de demodulación para obtener sus componentes YUV, el bloque de matrizado a la salida es opcional y, permite la obtención de señales convertidas a RGB. El decoder utilizado por nosotros es un decoder digital en el cual la señal de video compuesto es digitalizada y todas las

operaciones representadas en la figura son realizadas en dominio digital. La señal de salida resultante será un tren de muestras correspondiente a la señal en componentes en formato digital.

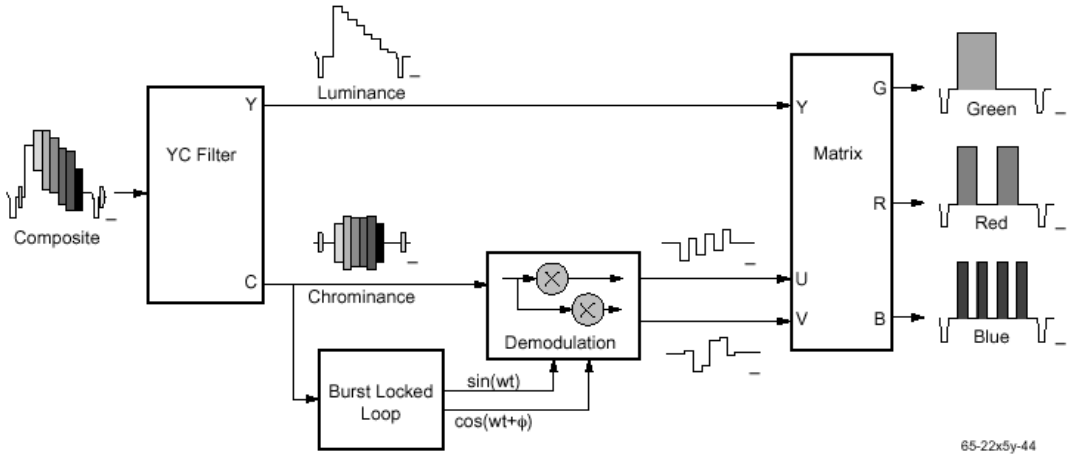


Figura 16. Diagrama de bloques de un decoder de vídeo.

La utilización de un decoder digital permite disponer de la señal demodulada en formato digital, mucho mas adecuada para realizar los procesos que posteriormente realizaremos en la FPGA.

Aparte de los bloques expuestos, un dispositivo de este tipo genera las correspondientes referencias de temporización o sincronismos y las incluye en la señal de salida. Estos dispositivos contienen, además, registros de configuración que permiten programar el funcionamiento del mismo dependiendo de las características de la señal de entrada, la programación se realiza escribiendo en determinados registros de configuración desde un controlador externo.

Este controlador será implementado en la FPGA, utilizando un interfaz de dos hilos por el que se realiza una comunicación serie con señales de reloj y datos. Este interfaz recibe el nombre de I2C.

El formato de muestras digitales de video que entrega el decoder, y que recibe la FPGA, para su posterior tratamiento, se denomina ITU-R BT.656, está normalizado por ITU (International Telecommunication Union) y define como se pueden integrar dentro de una trama digital información relevante tanto a la señal de video como a los sincronismos horizontales y verticales inherentes a la misma. Es necesario un módulo decodificador de dicha trama digital para poder procesar adecuadamente la señal de video, este módulo se implementa en la FPGA.

Información adicional sobre el bus I2C y el formato de señal ITU-R BT.656 se incluyen en los apartados siguientes.

2.4 Bus I2C.

El bus I2C consta de 2 hilos (reloj y datos) y permite una comunicación entre dos o más dispositivos por medio de un protocolo serie. Cada dispositivo conectado al bus tiene una dirección base, a la que responde con un ACK (Acknowledge) de reconocimiento de transmisión. Cada dispositivo puede tener varios registros internos que necesitaremos acceder para escribir o leer datos en ellos. La trama básica de comando I2C consta de 3 bytes. Una dirección base, una subdirección de registro y los datos correspondientes. Los comandos pueden ser de lectura o escritura pudiendo variar el formato de la trama en ligeros aspectos dependiendo del dispositivo que estemos controlando.

Las dos señales que conforman el bus se denominan SDA (para datos) y SCL para reloj. El estado de reposo del bus se da cuando los dos hilos se encuentran a nivel lógico alto o “1”. Cada transición a nivel bajo o “0” y posteriormente a “1” constituye un pulso de reloj y el valor lógico existente, en el flanco de subida de la señal SCL, es almacenado por los dispositivos colgados del bus.

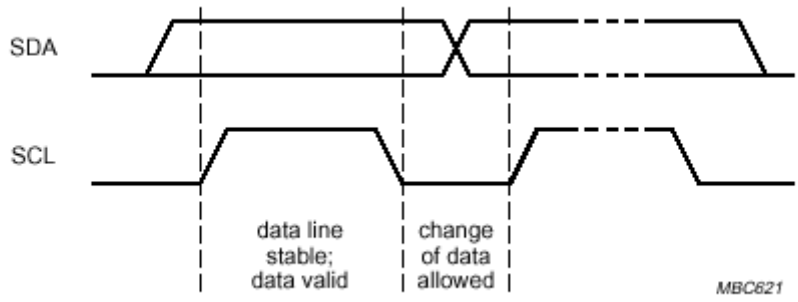


Figura 17. Transmisión de un bit en bus I2C.

La configuración del bus puede ser “monomaster” o “multimaster” dependiendo de, si existe un solo dispositivo que escribe en el bus o hay varios. En nuestro caso implementaremos un bus I2C con un solo maestro (FPGA) y un solo esclavo (decoder de video). La FPGA realizará la programación del decoder mediante este bus al iniciar el sistema, de modo que pueda decodificar la señal de video compuesto que apliquemos a su entrada.

El estado de reposo del bus se da cuando las dos líneas SDA y SCL se encuentran a nivel lógico alto. Al ser un bus con transmisión serie necesitamos un mecanismo de sincronización que nos permita identificar el comienzo y final de una transacción. Esto se consigue limitando, únicamente, las variaciones de nivel de la señal SDA cuando SCL se encuentra a “0”, es decir la variación de nivel de SDA cuando SCL se encuentra a “1” es interpretada por el controlador de bus I2C como un comando de operación o error de transmisión.

Partiendo del estado de reposo inicial se provoca el inicio de una transmisión realizando una transición de SDA desde “1” a “0” manteniendo SCL a “1”, es conocido como condición de “start”. La condición de “stop” se produce cuando existe una transición de “0” a “1” en SDA estando SCL a “1”.

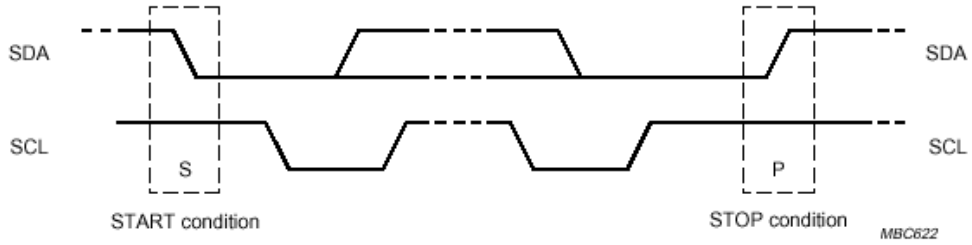


Figura 18. Condiciones “start” y “stop” en un bus I2C.

Cada byte escrito por el master en el bus debe recibir un ACK por parte del esclavo al que va dirigido como reconocimiento de operación, esta condición de ACK se da en el noveno pulso de SCL.

Así, y dependiendo de cada dispositivo particular, se consigue la transferencia de información y configuración de los distintos elementos conectados al bus. Las operaciones posibles son lectura y escritura de registros.

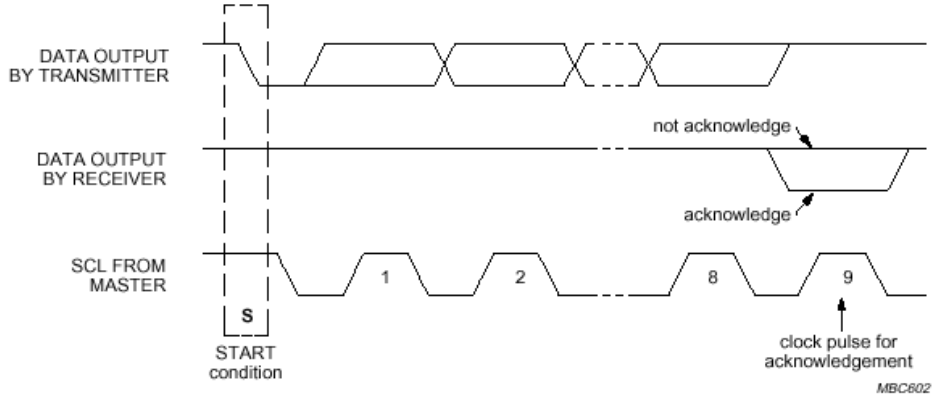


Figura 19. ACK en bus I2C.

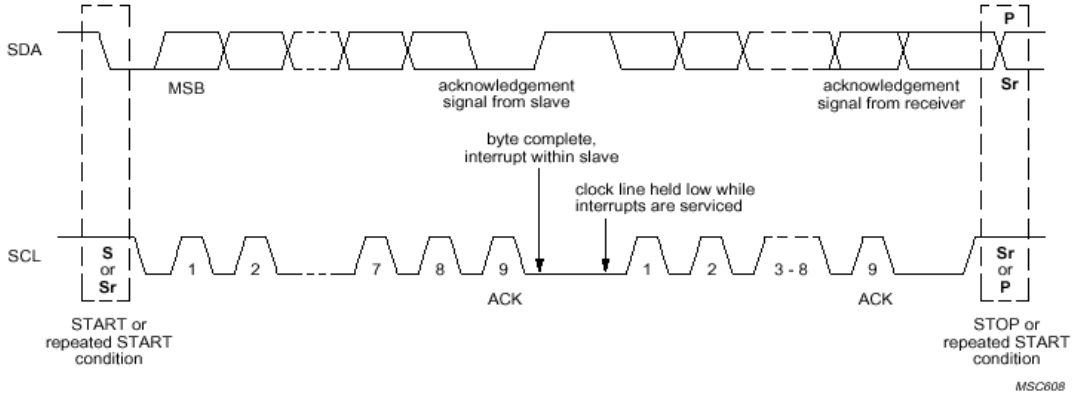


Figura 20. Transmisión de datos en un bus I2C.

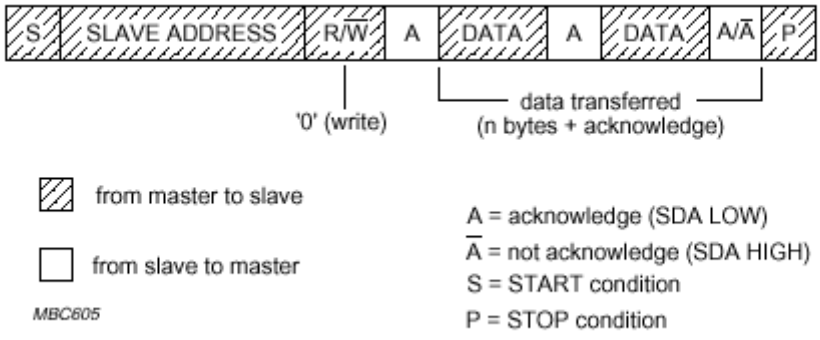


Figura 21. Operación de escritura.

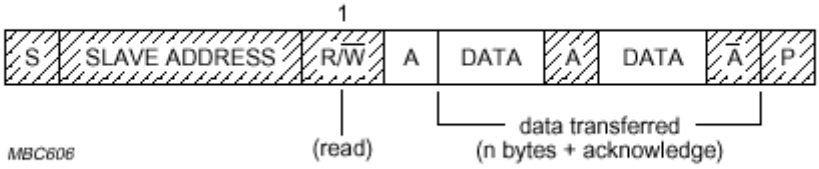


Figura 22. Operación de lectura

2.5 Memoria de imagen.

Para poder realizar las transformaciones requeridas a la señal de video necesitamos almacenar los diferentes campos de imagen constituyentes de la señal de video entrelazado y conseguir una señal con formato progresivo.

Las muestras correspondientes a la señal de video se almacenan en una memoria denominada "frame buffer", que en este caso se implementarán con un banco de 2 memorias SRAM de 512k x 8 bits. Dando un total de 512 x 16 bits, con bus de acceso de datos de 16 bits.

Los accesos a memoria son gestionados por la FPGA que debe intercalar accesos de lectura y escritura hacia la memoria convenientemente multiplexados para permitir un tratamiento en tiempo real de la señal de video. Debido a las restricciones temporales existentes entre dichos accesos existen ciertas limitaciones al diseño que comentaremos posteriormente.

Las memorias utilizadas son RAM estáticas con ciclos de acceso asíncronos. Un diagrama de bloques de las memorias utilizadas se presenta en la siguiente figura.

El array de memoria está controlado por decodificadores de fila y columna, nosotros aprovecharemos esta circunstancia para direccionar las filas con un contador de píxeles de línea y las columnas mediante un contador de líneas correspondiente al frame de vídeo. De este modo cada línea de vídeo se almacenará en una columna de la memoria y los píxeles individuales se controlan mediante accesos al decodificador de filas.

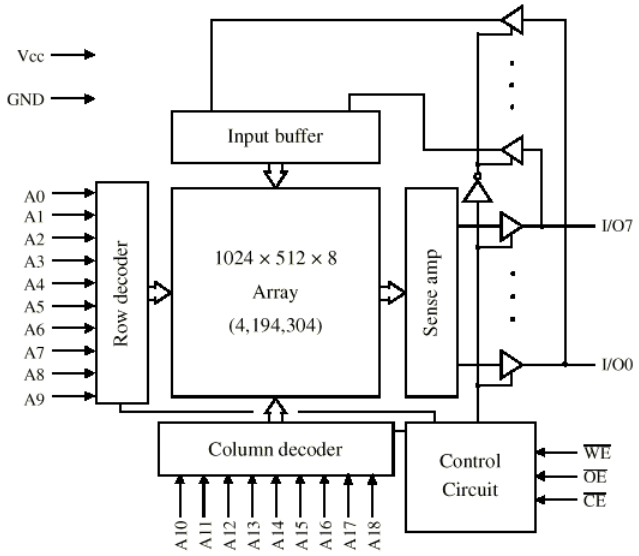


Figura 23. Arquitectura de la memoria utilizada.

2.6 RAMDAC.

Para poder visualizar la imagen en un CRT debemos generar señales analógicas correspondientes a las componentes RGB además de señales de sincronización horizontal vertical. Las señales de sincronismos H+V son sintetizadas en la FPGA, las muestras correspondientes a RGB necesitan ser convertidas mediante convertores D/A para excitar el tubo de imagen.

El dispositivo encargado de tal función en nuestro circuito es un RAMDAC, está constituido, básicamente, por tres convertidores D/A, uno para cada componente RGB, además de 3 memorias RAM de 256x8 bits.

El RAMDAC puede funcionar en dos modos principalmente, como convertidor D/A directo de las muestras de entrada, o indexando las memorias RAM a traves de las muestras de entrada para generar valores de conversión distintos. El modo de funcionamiento es programado desde la FPGA escribiendo en determinados registros del dispositivo.

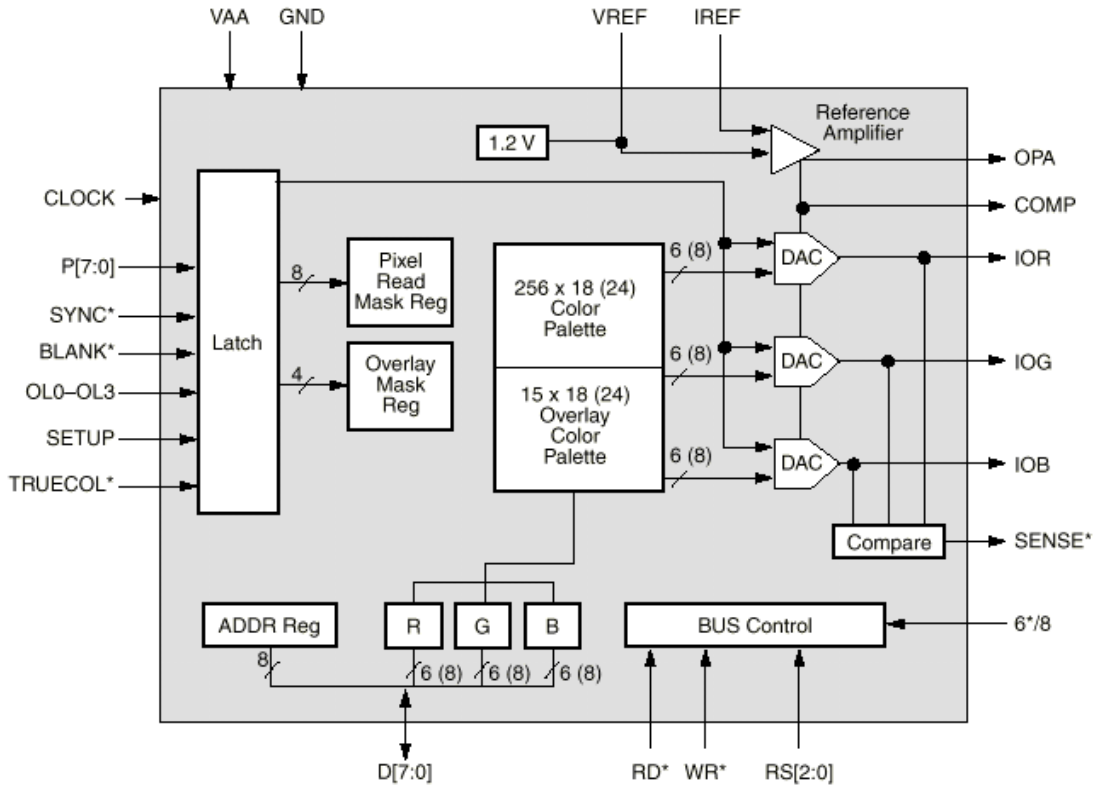


Figura 24. Arquitectura del RAMDAC utilizado.

En nuestro caso programamos el RAMDAC para que utilice las memorias RAM como tablas de conversión y generar los valores hacia los DAC para obtener las componentes analógicas de la señal de video. El contenido de las memorias es programado por la FPGA, cabe decir que dependiendo de los valores escritos en dichas memorias se pueden obtener diferentes señales analógicas en respuestas a las mismas muestras de entrada esto permite la generación de diferentes paletas de colores dependiendo de la aplicación.

En el diagrama de bloques se aprecian las memorias RAM internas y los tres convertidores D/A, para cada componente, existe un bloque permitir la configuración a través de un controlador externo y un puerto de entrada de muestras para convertir, las muestras entregadas al RAMDAC serán las generadas por la FPGA después de procesar el video de entrada.

2.7 Dispositivo de lógica programable (FPGA)

Una FPGA es un dispositivo de lógica programable diseñado para permitir construir sistemas digitales diversos dentro del mismo. Básicamente, consiste en una agrupación de puertas lógicas, flip-flops y redes de interconexión configurables, de modo que mediante un fichero de configuración descargado externamente podemos hacer que realice diferentes funciones. Permite la integración de todo tipo de módulos digitales tales como contadores, comparadores, multiplexores así como de maquinas de estados finitos. De esta forma, y mediante entorno de desarrollo apropiado utilizando un lenguaje de descripción hardware, como VHDL, podemos construir un bloque con el funcionamiento deseado.

La FPGA está constituida por una serie de bloques básicos como el presentado en la figura 22, mediante la adecuada interconexión de varios de estos bloques se puede implementar cualquier circuito lógico que especifiquemos.

En dicha FPGA se ejecutará el algoritmo de procesado de señal objeto de este Proyecto. Explicaciones mas detalladas sobre este algoritmo y su implementación se incluyen en los apartados siguientes de esta memoria.

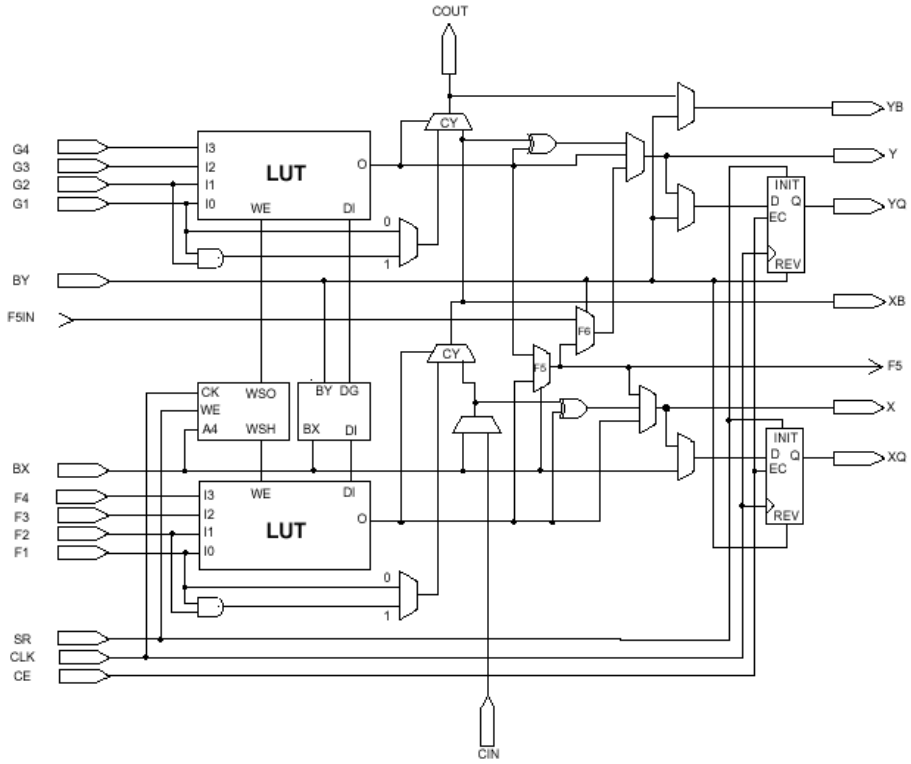


Figura 25. Bloque básico de FPGA.

3. ENTORNO DE DESARROLLO

Aunque el corazón de nuestro diseño es la construcción de un núcleo de procesamiento, codificado en VHDL, sintetizado y materializado sobre una FPGA, son necesarios otros elementos adicionales como un decoder de video, memoria RAM y un convertidor D/A como los introducidos previamente. Todos estos elementos se encuentran sobre una tarjeta de desarrollo denominada XSV800. Para poder construir nuestro núcleo VHDL que se implementará sobre la FPGA necesitamos un entorno de desarrollo que permita su codificación, depuración, compilación y síntesis. En concreto usamos una FPGA de 800.000 puertas de la familia Virtex de Xilinx. Dicho entorno se denomina Xilinx Foundation.

3.1 Placa prototipado.

La tarjeta utilizada en el desarrollo de este proyecto se denomina XSV800 y presenta un aspecto como el de la siguiente figura.

En la tarjeta se pueden apreciar los elementos anteriormente citados necesarios para el funcionamiento del sistema completo. La tarjeta contiene además otros elementos que permiten la proyección de diferentes diseños sobre la misma. Los video jacks en la parte superior derecha servirán para conectar nuestra fuente de video a convertir, el decoder de video realizará la correspondiente demodulación y conversión a digital de la señal, esta señal en formato digital llega a la FPGA, en el centro, que apoyado por las memorias SRAM aplicarán el algoritmo correspondiente. Finalmente el RAMDAC situado en la parte inferior derecha de la tarjeta realizará una conversión D/A y conectando un monitor de PC en el puerto VGA de salida podremos ver la señal de video correspondiente.

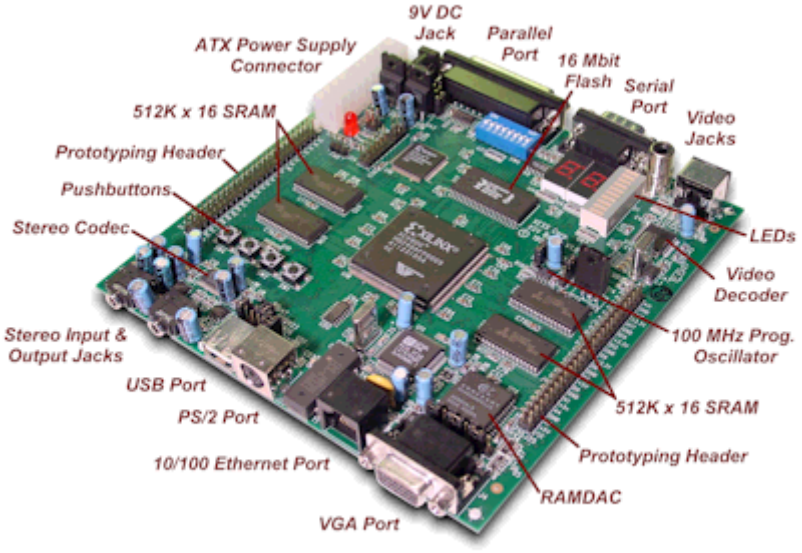


Figura 26. Placa prototipado XSV800.

En la siguiente figura se aprecia un diagrama de bloques de la tarjeta donde se pueden localizar los componentes utilizados encerrados por la línea discontinua.

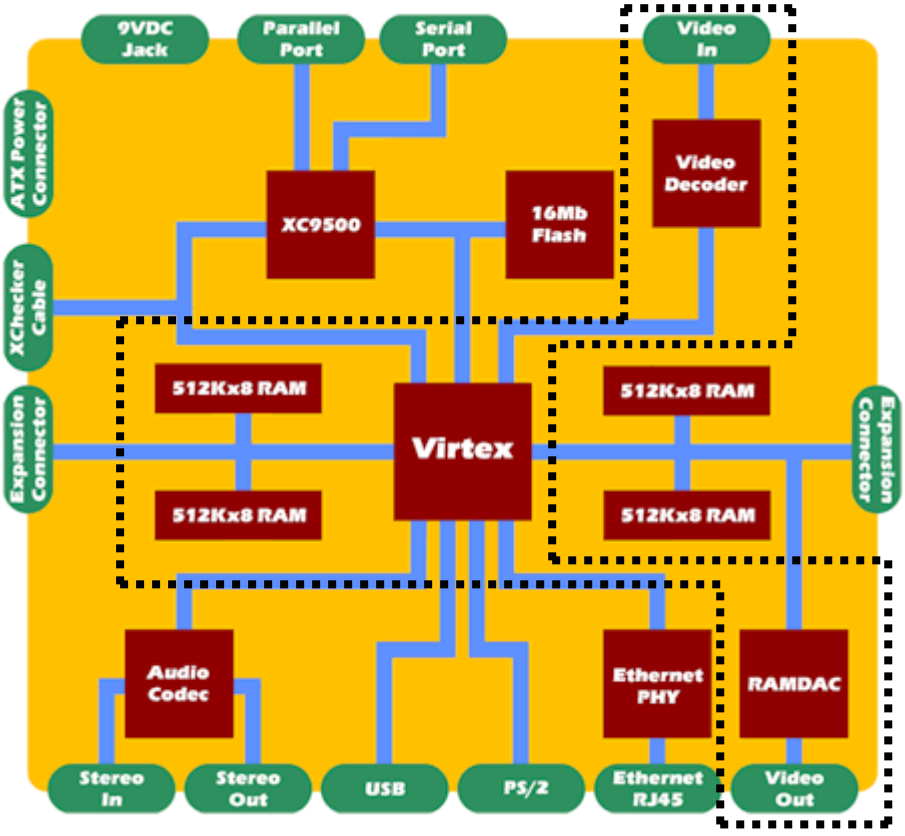


Figura 27. Diagrama de bloques de la tarjeta.

3.2 Software de síntesis

Nuestro trabajo consistirá en construir sobre la FPGA el sistema adecuado al procesamiento que queremos realizar. La lógica que sintetizemos dentro de ella definirá el funcionamiento del sistema, para este u otro proyecto. La herramienta que permite el desarrollo de esta lógica es un entorno de desarrollo software denominado Foundation de Xilinx, permite la síntesis lógica a partir de descripciones hardware de alto nivel por medio de lenguaje VHDL.

Dicho entorno permite la introducción del diseño como ficheros de texto dónde se realiza una descripción del circuito a sintetizar por medio de VHDL, la herramienta realiza entonces una compilación y síntesis de los elementos descritos y genera un fichero de configuración que se ejecutará una vez descargado sobre la FPGA.

El aspecto que presenta el entorno se puede ver en la siguiente figura.

En la figura se pueden apreciar las diferentes etapas por las que pasa un desarrollo con este tipo de herramientas.

- Introducción del diseño, dónde se crean los ficheros escritos en VHDL que conforman nuestro circuito.
- Etapa de síntesis, dónde al herramienta diseña una red lógica (formada por puertas y biestables) cuyo comportamiento es el especificado por los ficheros VHDL.
- Implementación o “Place & Route”, proyección tecnológica sobre los recursos de la FPGA de los elementos sintetizados.

Paralelamente con este ciclo de diseño existen etapas de simulación y verificación de nuestro diseño, para ello se utilizan simuladores de lenguaje VHDL. Puede utilizarse el propio simulador que integra la herramienta Foundation o utilizar otro externo. En nuestro caso utilizamos Modelsim de Mentor Graphics, un simulador más potente y con más capacidades de verificación y depuración. Un ejemplo de diagrama temporal de una parte de nuestro diseño puede verse en la siguiente figura.

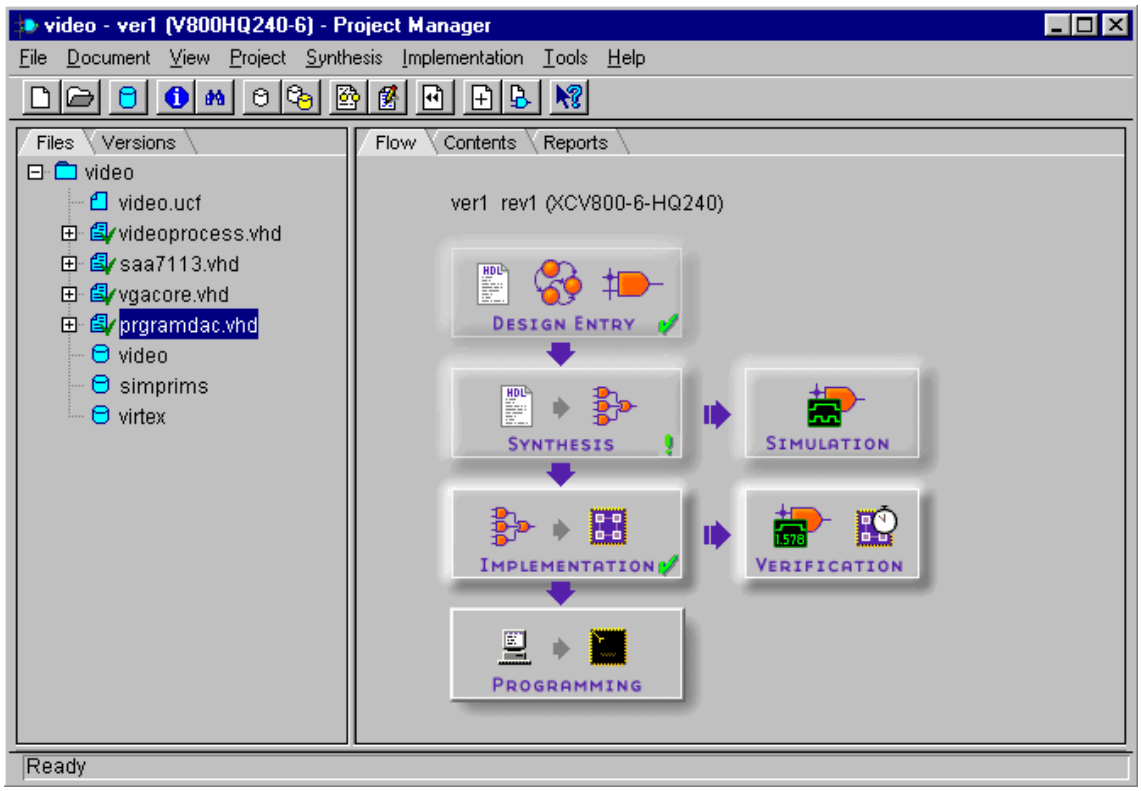


Figura 28. Entorno de desarrollo Foundation de Xilinx.

3.3 Lenguaje de especificación: VHDL.

El significado de las siglas VHDL es VHSIC (Very High Speed Integrated Circuit Hardware Description Language), es decir, lengiaje de descripción hardaware de circuitos integrados de muy alta velocidad. VHDL es un lenguaje de descripción y modelado diseñado para describir, en una forma en que los humanos y las máquinas puedan leer y entender la funcionalidad y organización de sistemas hardware digitales.

VHDL fue desarrollado como un lenguaje para el modelado y simulación lógica dirigida por eventos de sistemas digitales, y actualmente se utiliza también para la síntesis automatica de circuitos.
VHDL es un lenguaje con una sintaxis amplia y flexible que permite el modelado estructural, en flujo de datos y de comportamiento hardware. VHDL permite el modelado preciso, en distintos estilos, del comportamiento de un sistema digital conocido y el desarrollo de modelos de simulación.

Otro de los usos de este lenguaje es la síntesis automática de circuitos. En el proceso de síntesis se parte de una especificación de entrada con un determinado nivel de abstracción y se llega a una implementación más detallada, menos abstracta. Por tanto, la síntesis es una tarea vertical entre niveles de abstracción, del nivel más alto en la jerarquía de diseño hacia el más bajo nivel de jerarquía.

VHDL es un lenguaje que fue diseñado inicialmente para ser usado en el modelado de sistemas digitales. Es por esta razón que su utilización en síntesis no es inmediata, aunque lo cierto es que la sofisticación de las actuales herramientas de síntesis es tal que permiten implementar diseños especificados en un alto nivel de abstracción.

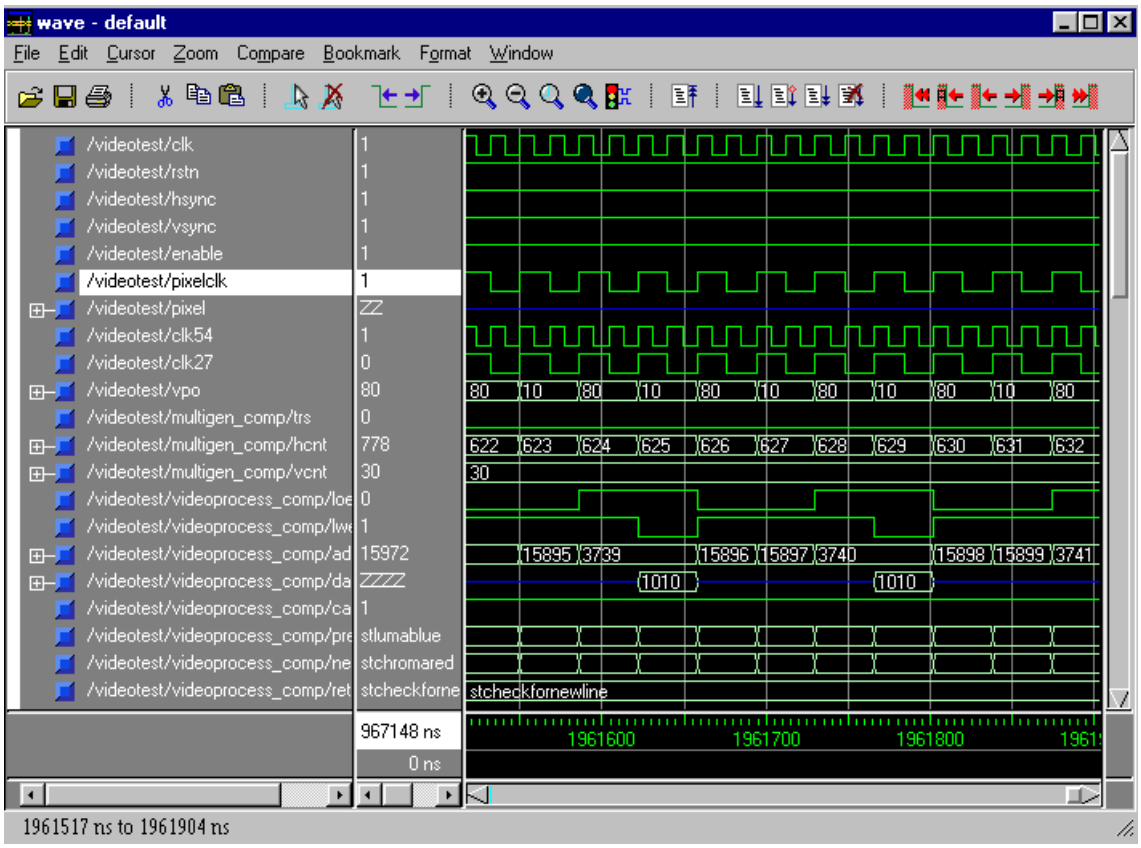


Figura 29. Forma de onda procedente de Modelsim.

Algunas ventajas del uso de VHDL para la descripción hardware son:

- VHDL permite diseñar, modelar y comprobar un sistema desde un alto nivel de abstracción bajando hasta el nivel de definición estructural de puertas.

- Circuitos descritos utilizando VHDL, siguiendo unas guías de síntesis, pueden ser utilizados por diversas herramientas de síntesis para crear e implementar circuitos.
- Los módulos creados en VHDL pueden utilizarse en diferentes dfiseños, lo que permite la reutilización de código. Además, la misma desripción puede utilizarse para diferentes tecnologías sin tener que rediseñar todo el circuito.
- VHDL permite diseño Top-Down, esto es, describir (modelar) el comportamiento de los bloques de alto nivel, analizarlos (simularlos) y refinar la funcionalidad en alto nivel requerida antes de llegar a niveles mas bajos de abstracción de la implementación del diseño.
- Modularidad: VHDL permite dividir o descomponer un diseño hardware y su descripción VHDL en unidades mas pequeñas.

VHDL permite, por tanto, implementar un diseño completo en base a módulos más pequeños. También permite la creación de diseños jerárquicos desde que permite la creación e instanciación de componentes en las descripciones VHDL.

Un componente en un módulo descrito en VHDL que tiene una ENTIDAD y ARQUITECTURA propias. Una ENTIDAD es un elemento VHDL que define el interfaz de un determinado circuito y especifica el número y tipo de los puertos de entrada/salida que dispone. La ARQUITECTURA define el comportamiento que presenta el circuito.

4. DISEÑO E IMPLEMENTACIÓN

4.1 Consideraciones iniciales.

En este apartado se discute el flujo de diseño seguido hasta la realización del sistema completo. Partiendo de las características del hardware disponible y las herramientas utilizadas para la proyección del diseño sobre FPGA son necesarias ciertas consideraciones iniciales que repercutirán en las soluciones adoptadas.

Se pretende realizar un sistema que tome como entrada una señal de video compuesto estándar de cualquier fuente como puede ser un reproductor de video, una videocámara o la salida de un sintonizador de TV por poner unos ejemplos. La salida de nuestro sistema deberá presentar las características apropiadas para poder ser capturada y tratada por un ordenador personal o ser visualizada en un monitor de PC.

Las características más relevantes de una señal de video compuesto estándar estriban en que la información de color viene modulada con una portadora y sus barridos de sincronización son entrelazados como se ha comentado anteriormente.

Una señal apta para su tratamiento por PC debe mantener información en espacio de color RGB y mantener un barrido progresivo. Así pues, el objetivo de nuestro proyecto será transformar una señal de 625 líneas 50Hz, barrido entrelazado y modulación PAL de la información de color en una señal de 625 líneas 50Hz, barrido progresivo y en componentes de color RGB.

Con estas consideraciones presentes se hace un estudio de la topología que presenta la placa de prototipado XSV800 que constituirá el hardware que soporte nuestro diseño.

La placa dispone de 2 bancos de memoria de 512k x 16 bits, denominados derecho e izquierdo. Uno de los bancos, concretamente el derecho comparte lineas del bus con el dispositivo RAMDAC lo que hace inutilizable este banco para procesado de vídeo. Con las memorias disponibles únicamente tenemos capacidad de transferencia para realizar una conversión monocroma. Es decir, la señal digital en componentes entregada por el decoder de video a la FPGA no podrá ser manipulada en tiempo real para obtener una señal RGB en color a la salida de nuestro sistema, una solución de compromiso implica el tratamiento de la señal en escala de grises de modo que las necesidades de memoria se reducen a la mitad al desechar las muestras correspondientes a las componentes B-Y, R-Y.

En cuanto al proceso de desentrelazado, se aplica un algoritmo de replicación de líneas de modo que una línea de un campo de vídeo es presentado en dos líneas consecutivas, de modo que se condigue una conversión a formato progresivo con actualización de refresco de 50 Hz, doble que la señal original que traza una imagen completa en 2 pasadas.

Se implementa, adicionalmente, una utilidad de congelado de imagen que permite, mediante la pulsación de un interruptor situado en la tarjeta, deshabilitar la captura de video de modo que el refresco de la señal progresiva mantiene el último frame capturado, soltando el botón se vuelve al método de captura original y la señal de vídeo vuelve a ser presentada en pantalla.

Sabiendo que el tren de muestras entregadas por el decoder de video contiene información de video en espacio de color Y,B-Y,R-Y, nos es necesario únicamente manejar las muestras correspondientes a luminancia (Y), necesitamos configurar un mapa de colores adecuado en las memorias del RAMDAC. En un espacio de color RGB se obtiene una escala de grises cuando se igualan los valores de las componentes RGB de una señal, recordar que una mezcla de colores aditivos rojo, verde y azul forma un blanco. Dependiendo de la intensidad de las señales que generemos se podrá crear una escala de grises. Como trabajamos con el RAMDAC en modo 8 bits por píxel utilizando las tablas de conversión podemos disponer de una señal resultante en una escala de 256 niveles de gris.

El hardware sintetizado sobre la FPGA se encarga de realizar una inicialización de dispositivos externos (decoder y RAMDAC) y decodificar la señal procedente del decoder, extraer de la misma la información relativa a la señal de video, valiéndose de los TRS inmersos en la trama y almacenando en memoria las muestras correspondientes a la luminancia de la señal. También se integra un controlador VGA que genera las señales de sincronismo horizontal y vertical adecuada para un monitor de PC con barrido progresivo y se leen muestras de memoria correspondientes a la posición de pantalla correspondiente. Estas muestras son enviadas al RAMDAC para su conversión analógica y visualización en el monitor. Un controlador gestiona los accesos a memoria para lectura y escritura y así evitar conflictos en el bus.

Es necesario decodificar bien la información de temporización embebida en la señal de video para conseguir un centrado de la imagen en el monitor VGA, debemos localizar la primera línea de la imagen de video de entrada y escribirla en una posición de memoria correspondiente con las direcciones de lectura para el refresco de la primera línea del controlador VGA y así sucesivamente.

4.2 Diseño.

Nuestro diseño consistirá en la creación de un núcleo hardware de procesamiento que, descargado en la FPGA, permita a la tarjeta XSV800 realizar las funciones anteriormente citadas. Utilizando como lenguaje de diseño VHDL y, valiéndonos de las herramientas descritas, codificamos los módulos correspondientes al sistema.

4.2.1 Partición del diseño

La FPGA contiene una serie de pines de entrada/salida que permiten la comunicación del diseño implementado en su interior con otros dispositivos externos. Los pines necesarios serán asignados de acuerdo a los esquemáticos de la tarjeta XSV800 que se incluyen en los apéndices. El sistema completo integrado describirá un funcionamiento que entra en comunicación con el resto de la tarjeta a través de un interfaz de puertos que irán conectados a los pines correspondientes.

El sistema implementado consta de una serie de módulos de forma que se genera una jerarquía de componentes que se detalla a continuación.

El nivel mas alto de jerarquía lo forma el módulo SAA7113, este módulo integra un componente de cada elemento de la jerarquía inmediatamente inferior. Así SAA7113 integrará un componente debouncer, prgramdacver2, i2c y videoprocess. El módulo videoprocess contiene a su vez un componente de jerarquía inferior llamado vgacore.

Cada módulo se desarrolla en un fichero VHDL distinto con el mismo nombre del módulo. A continuación se listan los ficheros que constituyen el sistema, una descripción de los mismos y la función que desempeñan.

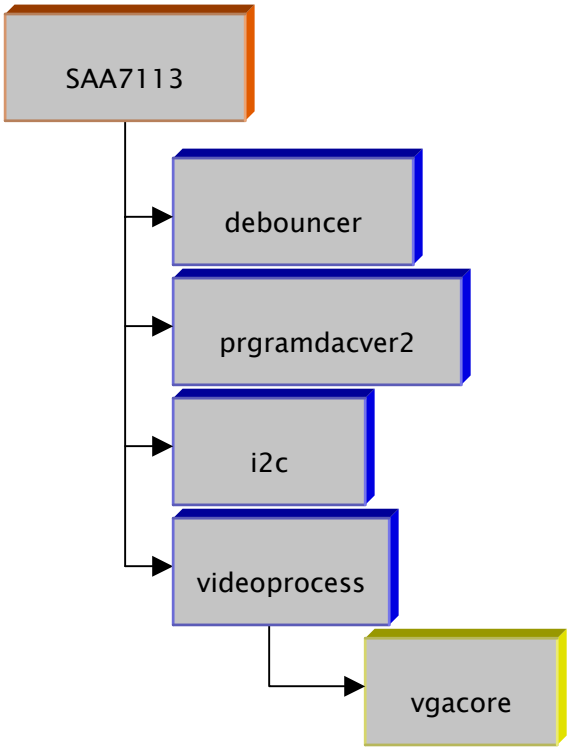


Figura 30. Jerarquía de módulos en el sistema.

- **Vgacore.vhd**

El módulo genera los pulsos de sincronismo adecuados para el monitor. Genera los pulsos de sincronismo H y V, implementa un contador de pixeles y un contador de líneas que sirven al módulo que lo contiene (videoprocess) para generar las direcciones de lectura de memoria adecuadas a la posición que se está procesando. Genera además una señal de borrado hacia el RAMDAC para llevar a nivel de negro la señal de video en zonas de sincronismo tanto horizontal como vertical.

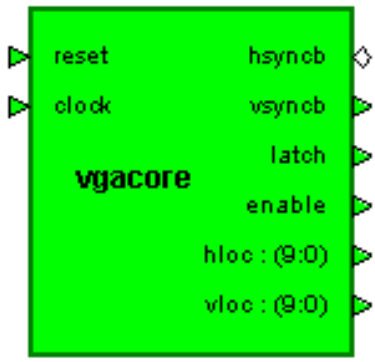


Figura 31. Simbolo del módulo vgacore.vhd

Los puertos de entrada/salida del módulo son:

Clock: Entrada de reloj del módulo, 27 MHz.
Reset: Entrada activa a nivel bajo, deja el módulo en un estado inicial.
Hsyncb: Señal bidireccional, proporciona los pulsos de sincronismo horizontal hacia el monitor
Vsyncb: Salida de sincronismo vertical.
Enable: Señal hacia el RAMDAC que provoca el borrado en zonas de sincronismo. Las salidas del RAMDAC son llevadas a nivel de negoro cuando esta señal tiene un nivel bajo.
Hloc(9..0): Contador horizontal de píxeles.

Vloc(9..0): Contador vertical de líneas. Estos dos buses son utilizados para generar las direcciones de lectura a la memoria de refresco.

- **Videoprocess.vhd**

Este módulo contiene un componente vgacore y realiza el procesado de la señal de video. Recibe el video procedente del decoder, lo procesa extrayendo los TRS y las muestras de luminancia para almacenarlas en memoria. Gestiona los accesos a memoria para lectura y escritura. Las direcciones de acceso de escritura son formadas según información de frame obtenida a través de TRS de la señal de video, indexando la memoria según píxeles y líneas procedentes de la misma. Las direcciones de lectura son formadas con la información entregada por los contadores del módulo vgacore.vhd.

Clk27: señal de reloj del módulo, 27MHz procedente del decoder de video SAA7113
Vpo(7..0): datos de video del decoder, estos datos de video corresponden a la señal de video compuesto que procesa el decoder.
Rstn: señal activa a nivel bajo que deja al sistema en un estado inicial
Capture: señal que proviene del módulo debouncer.vhd, es la versión filtrada de rebotes de un switch de la tarjeta, cuando capture está a nivel bajo se detiene la captura y la imagen es congelada.
Datamem(15..0): bus de 16 bits bidireccional que transfiere datos con el banco de memoria SRAM.
Pixel(7..0): datos hacia el RAMDAC que constituyen la señal de video procesada. Estas muestras actualizadas a reloj de pixelclk están listas para su conversión a analógica y presentadas en el monitor.
Pixelclk: reloj de pixel (27MHz) hacia el RAMDAC.
Enable: señal hacia el RAMDAC para borrar la señal de video en zonas de sincronismo, viene del módulo vgacore.
Lceb: señal de habilitación del banco izquierdo de memoria, la memoria se activa a nivel bajo.
Loeb: señal de habilitación de lectura de las memorias, esta señal será activada en ciclos de lectura.
Lweb: señal de habilitación de escritura de memoria, la señal se activa en ciclos de escritura.
Addmem(18..0): bus de direcciones del banco de memoria SRAM.
Capturando: señal que indica cuando el módulo está capturando señal de video, va conectado a un led de la tarjeta.

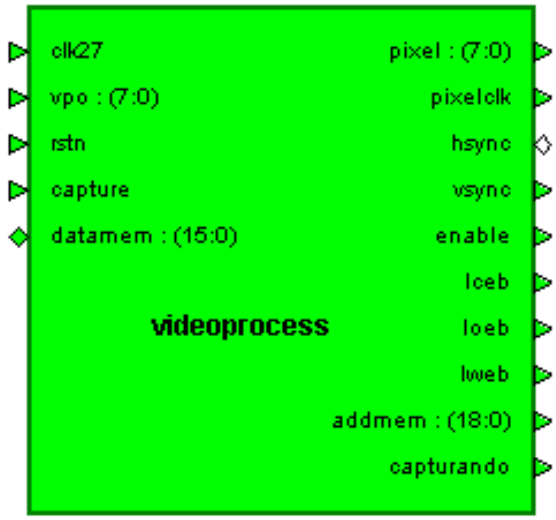


Figura 32. Símbolo del módulo videoprocess.vhd

- **I2c.vhd**

Este módulo realiza la programación inicial de los registros del decoder de vídeo SAA7113. Despues de un reset del sistema se sucede una programación del decoder por medio de este módulo.

Clk: entrada de reloj del módulo, usará el reloj global de la tarjeta 50MHz, el módulo implementa internamente un divisor de frecuencia para obtener un reloj con mayor período.
Rst: entrada de reset asíncrono del módulo, deja el módulo en condiciones iniciales.
Start: señal que, a nivel alto, provoca el inicio de la secuencia de configuración del decoder de video.
Done: señal que indica, a nivel alto, que la secuencia de configuración ha finalizado.
Scl: señal de reloj del bus i2c que configura el decoder.
Sda: señal de datos del bus i2c que configura el decoder.

Figura 33. Símbolo del módulo i2c.vhd.

- **Prgramdacver2.vhd**

Módulo que realiza la configuración del RAMDAC, el dispositivo tiene un puerto paralelo de configuración con buses de datos, direcciones y control. La configuración del RAMDAC también se sucede al iniciar el sistema después de un reset.

Figura 34. Símbolo del módulo prgramdacver2.vhd

Clk: Señal de reloj del módulo, usa el reloj global de la placa 50MHz, el módulo implementa un divisor de frecuencia interno.
Rstn: entrada de reset a nivel bajo.
Start: señal que activa la secuencia de configuración del RAMDAC.
RS(2..0): bus de selección de registro de configuración.
Data(7..0): bus de datos de configuración del RAMDAC
Done: señal que indica, a nivel alto, que la secuencia de configuración ha terminado.
WRn: señal de escritura del puerto de configuración del RAMDAC.
RDn: señal de lectura del puerto de configuración del RAMDAC.

- **Debouncer.vhd**

El módulo debouncer realiza un filtrado de rebotes del pulsador que controla la captura de vídeo para congelar la imagen. Consta de un temporizador que arranca cuando se pulsa o despulsa la tecla y tiene como salida una señal de captura sin oscilaciones.

Rst: Señal de reset asíncrono. Activo a nivel bajo.
Clk: señal de reloj. Utiliza el reloj de 27MHz del video. Implementa un divisor de frecuencias interno al módulo.
X: señal que proviene del switch de la placa.
Xdeb: señal de salida libre de rebotes hacia el módulo videoprocess para controlar la captura
Xdebfallingedge: señal que indica que se ha pulsado la tecla.
Xdebrisingedge: señal que indica que se despulsa la tecla.

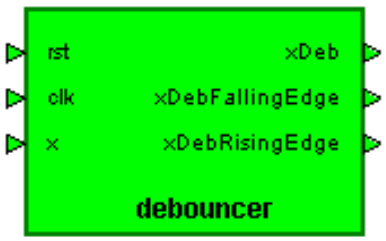


Figura 35. Símbolo del módulo debouncer.vhd

- **Saa7113.vhd**

Módulo de más alto nivel de jerarquía, integra un componente de los módulos descritos anteriormente (según la figura 29) y describe la interconexión entre sus puertos. Contiene tambíen una máquina de estados que controla la secuencia de incialización del sistema. Lanza las señales de comienzo a los módulos i2c.vhd y prgramdacver2.vhd y videoprocess.

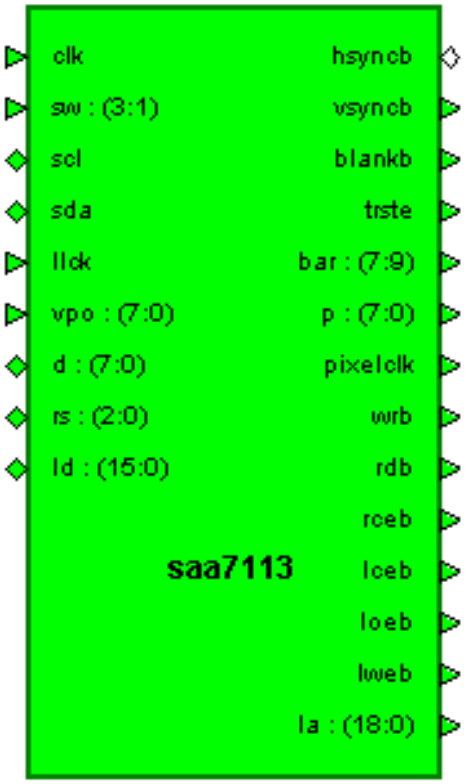


Figura 36. Símbolo del módulo saa7113.vhd

Clk: señal de reloj global del sistema. 50MHz de la tarjeta.
Sw(3..1): puertos de entrada de switches tales como reset y captura.
Scl: señal de reloj del bus i2c.

Sda: señal de datos del bus i2c.
Llck: reloj de video procedente del decoder (27Mhz) se utiliza fundamentalmente en el módulo videoprocess.
Vpo(7..0): datos de video procedentes del decoder.
D(7..0): bus de datos de configuracion del RAMDAC.
RS(2..0): bus de selección de registro de RAMDAC.
Ld(15..0): bus de datos bidireccional del banco de memoria SRAM.
Hsyncb: señal de sincronismo horizontal.
Vsyncb: señal de sincronismo vertical.
Blankb: Señal de control de borrado para el RAMDAC.
Trste: señal que inhabilita el controlador de ethernet que existe en la tarjeta para evitar confilctos con otros buses. Activo a nivel alto.
Bar(7..9): leds de la tarjeta para indicar diversa información como reset o estado de captura.
P(7..0): píxeles hacía el RAMDAC, serán los datos de la señal de vídeo a representar.
Pixelclk: reloj de conversión de píxel hacia el RAMDAC (27 MHz).
Wrb: señal de control de escritura del puerto de configuración del RAMDAC.
Rdb: señal de control de lectura del puerto de configuración del RAMDAC.
Rceb: señal de inhabilitación del banco derecho de memoria SRAM (no usado) comparte buses con RAMDAC. A nivel alto para inhibir.
Lceb: señal de inhabilitación del banco izquierdo de memoria SRAM (el utilizado para el procesado de vídeo). A nivel bajo para habilitar.
Loeb: señal de habilitación de lectura de memoria, se activa en ciclos de lectura.
Lweb: señal de habilitación de escritura de memoria, se activa en ciclos de escritura.
La(18..0): bus de direcciones del banco de memoria SRAM.

A continuación se representa un diagrama de bloques interno del módulo SAA7113, se puede apreciar que el módulo más alto de la jerarquía se compone de instancias de otros módulos y una interconexión entre ellos. Las señales con puerto de entrada/salida se corresponden con pines externos del diseño. Las señales de los módulos marcados con un circulo indica una conexión de señal interna al módulo.

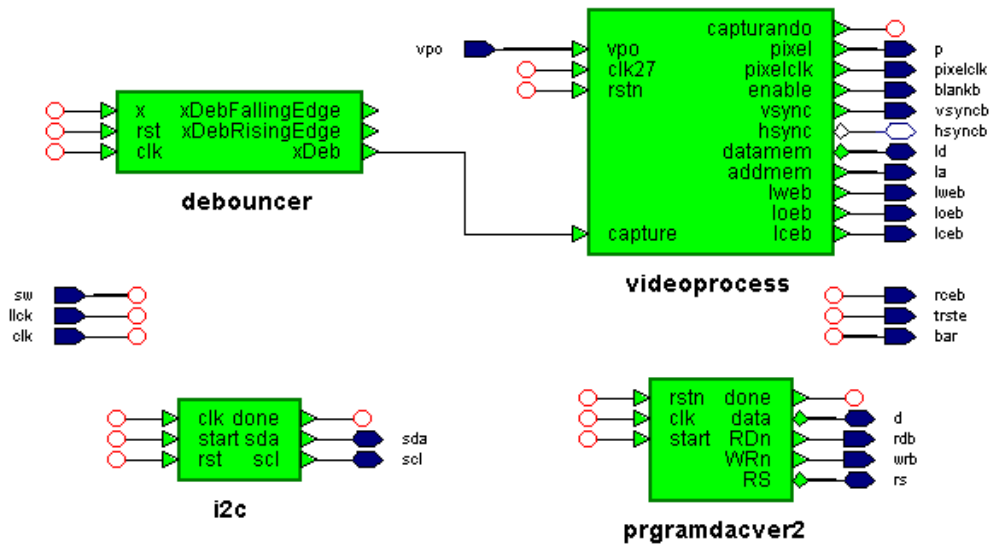


Figura 37. Diagrama de bloques interno del módulo saa7113.

4.2.2 Diagramas RT de los módulos.

A continuación se describen los módulos constituyentes del diseño con mayor profundidad, se especifica una descripción RT de cada uno de los módulos. Los listados con el código completo de cada uno de los módulos se incluyen en un apéndice al final de la memoria.

4.2.2.1 Debouncer.vhd

El módulo debouncer consta básicamente de 3 bloques, sincronizador, temporizador y controlador.

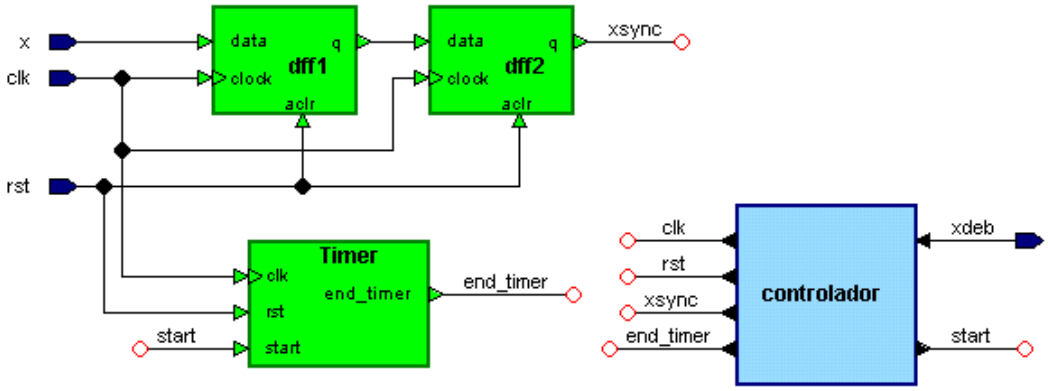


Figura 38. Diagrama RT del módulo debouncer.

El sincronizador está formado por 2 DFF que capturan la señal asíncrona x proviniente del pulsador y la convierten a una señal síncrona con el reloj. El controlador arranca un timer y espera a su finalización para dar como salida xDeb que es la versión filtrada de la señal del pulsador.

A continuación se muestra el diagrama de estados del correspondiente controlador.

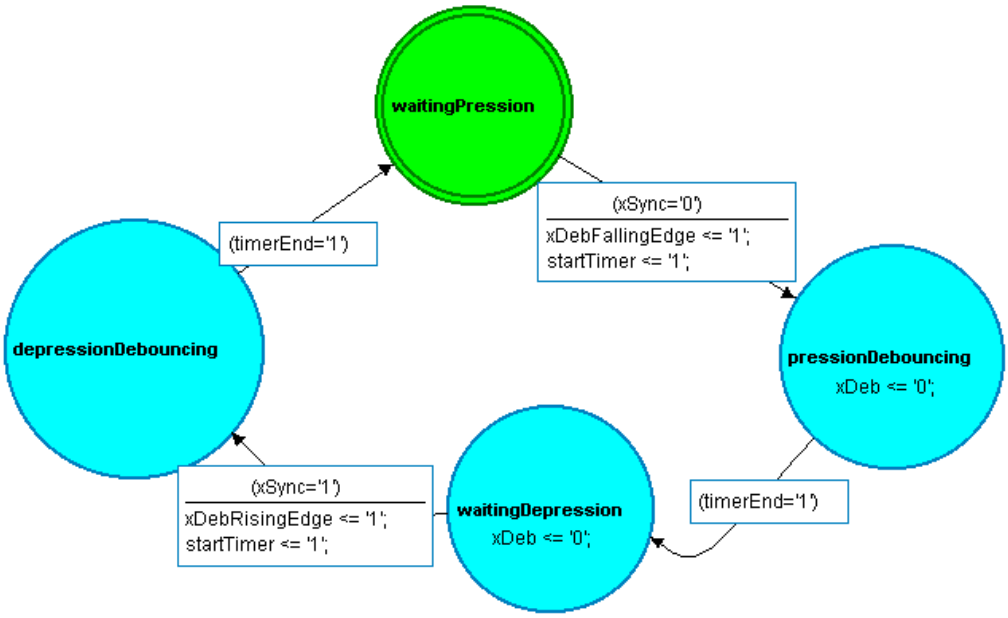


Figura 39. Diagrama de estados del controlador debouncer.

La señal Xdeb se mantiene por defecto a nivel alto. Cuando Xsync vale '0' se arranca el temporizador y Xdeb pasa a valer '0', cuando finaliza el periodo de cuenta del timer se espera a despulsar el switch, cuando esto sucede se vuelve a poner Xdeb a '1' y se arranca el timer. Cuando este finaliza estamos en posición para tratar una nueva pulsación.

4.2.2.2 I2c.vhd.

Este modulo se encarga de la inicialización del decoder de video SAA7113. Consta, principalmente de 2 memorias ROM donde se almacenan las direcciones y datos de los registros a escribir respectivamente. Adicionalmente es necesario un bloque que divide la frecuencia del reloj entrante, unos contadores de palabras y bits escritos, un registro de desplazamiento ya que la transmisión de información es serie y un controlador implementado como una maquina de estados finitos.

El controlador detecta la transición en la señal start y comienza la configuración, el reloj del controlador viene del divisor de frecuencias. Las memorias ROM son indexadas según la cuenta de palabras que se escriben. El registro de desplazamiento es cargado con la dirección o el dato según el multiplexor y se desplazan los bits. La salida SDA y SCL en última instancia están gestionadas por el controlador.

A continuación se muestra un diagrama de estados del controlador I2C. El diagrama se ha representado con niveles jerárquicos por ser bastante elevado el número de estados. Posteriormente se muestran los niveles mas bajos.

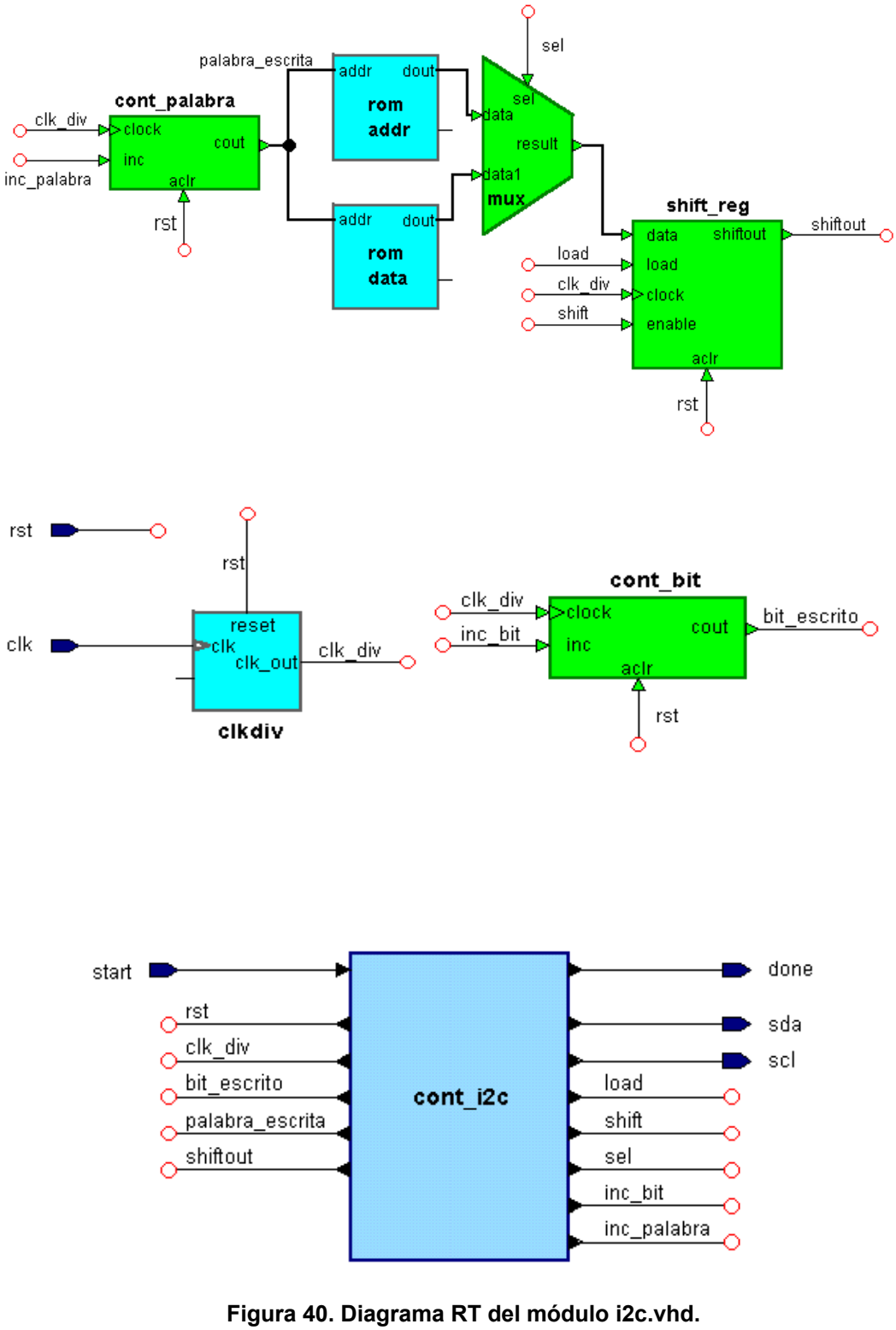


Figura 40. Diagrama RT del módulo i2c.vhd.

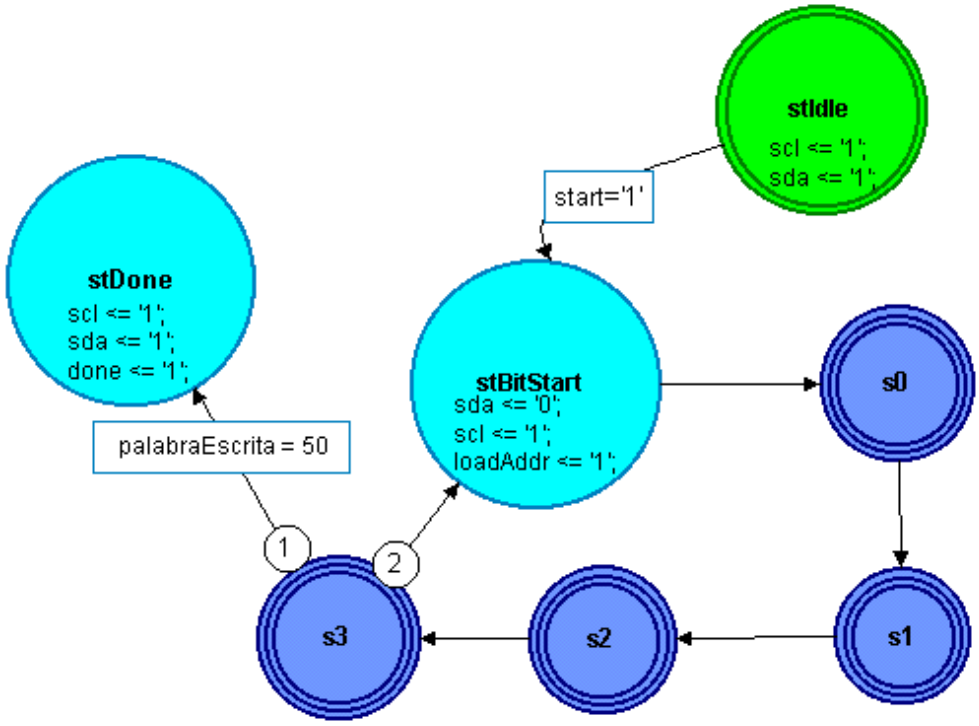


Figura 41. Diagrama de estados del controlador I2C.

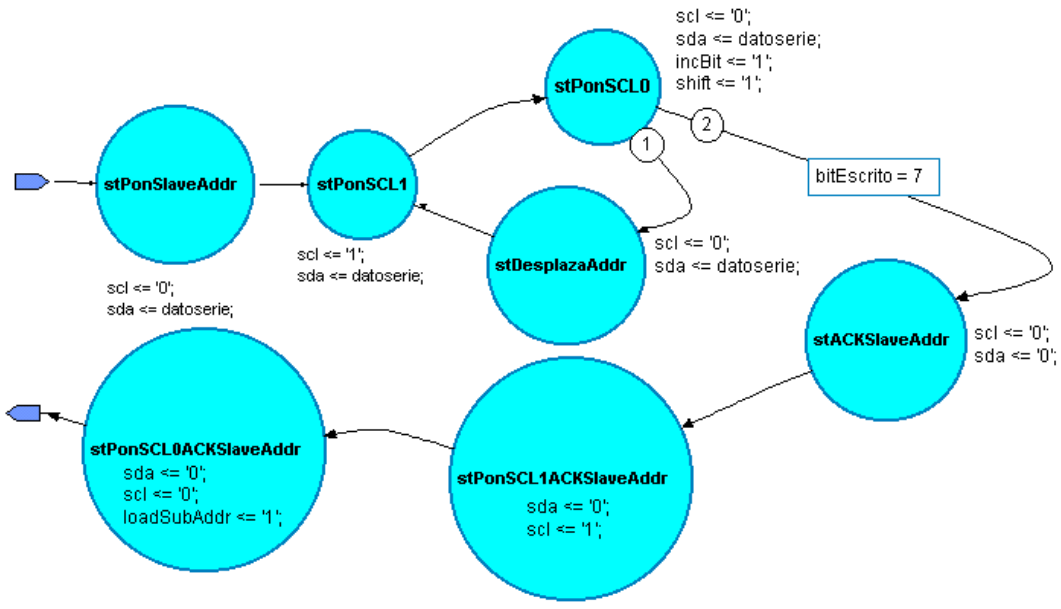


Figura 42. Estados dentro de S0.

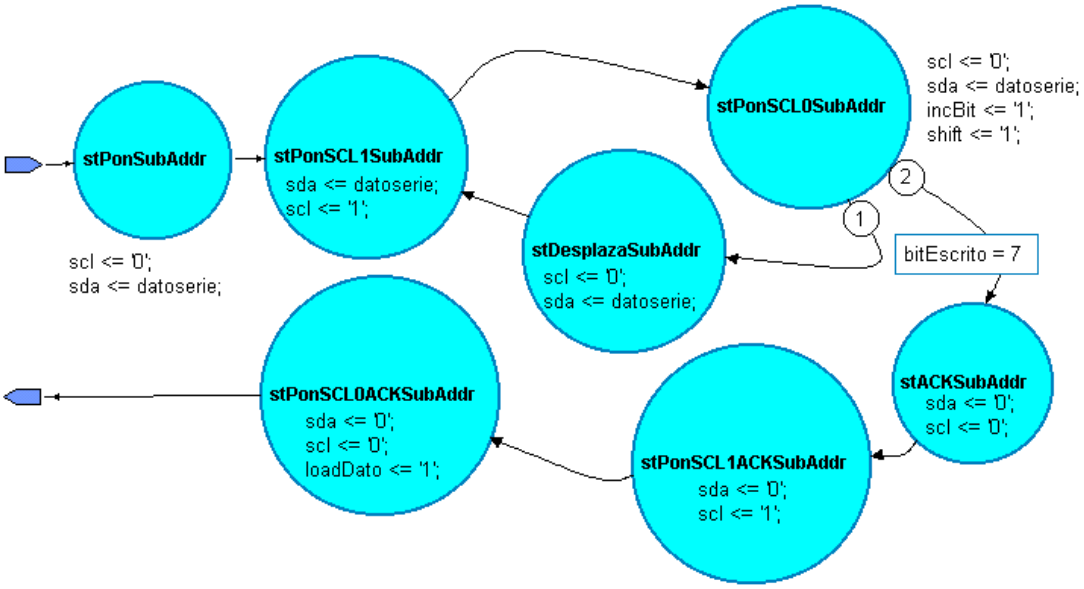


Figura 43. Estados de S1.

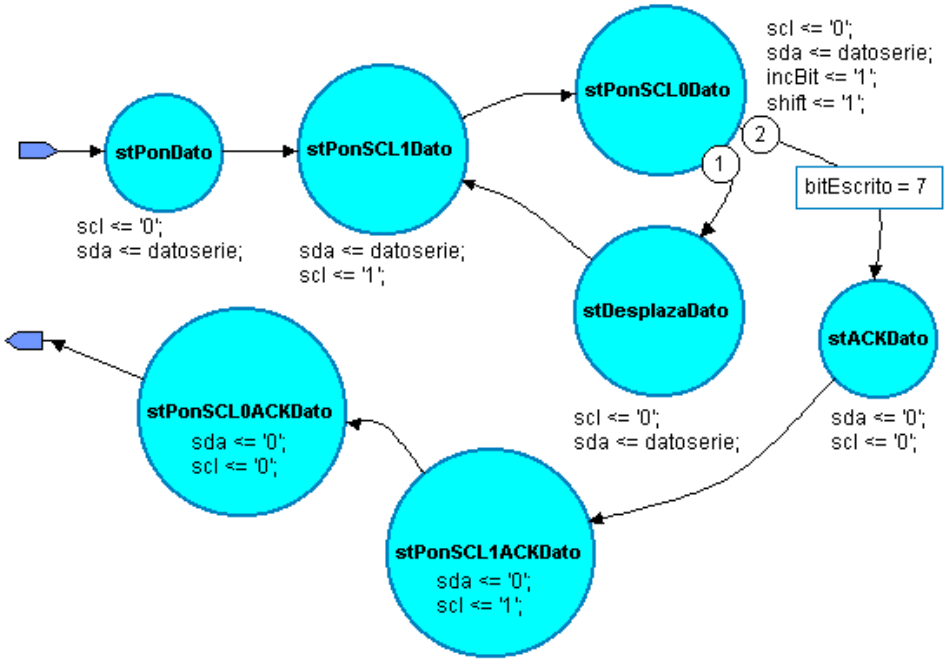


Figura 44. Estados de S2.

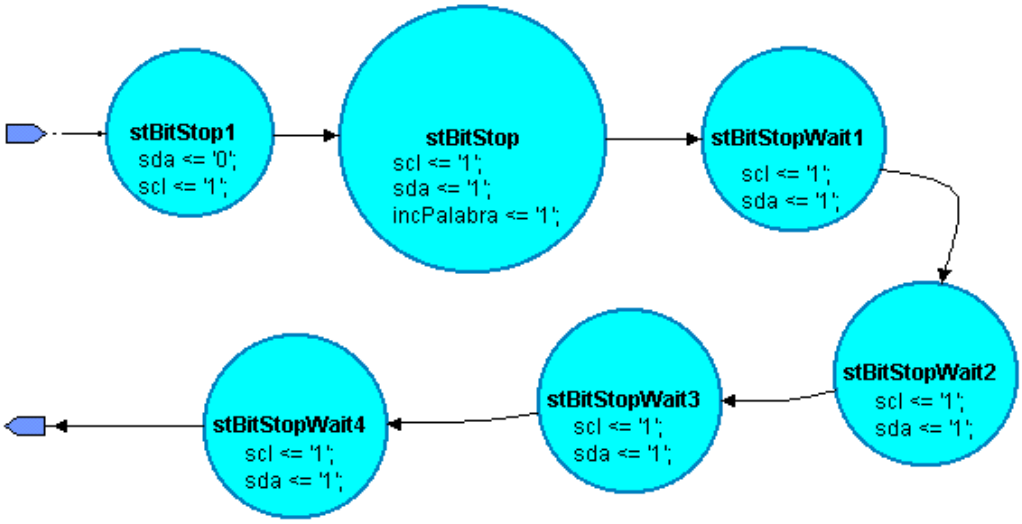


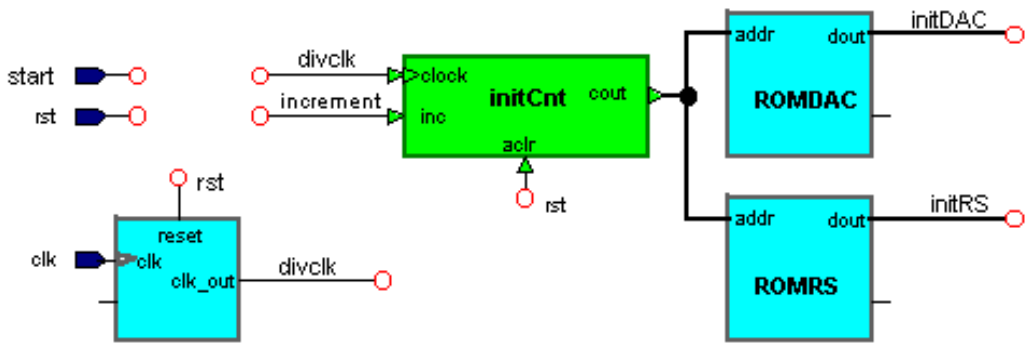
Figura 45. Estados de S3.

En la secuencia de figuras se aprecian los estados completos del controlador. En la figura 40 se presenta el nivel de jerarquía mas alto de la máquina. En las siguientes figuras se presentan los estados que comprenden cada nivel, básicamente S0 especifica la escritura de la dirección base del dispositivo, S1 la escritura de la dirección del registro al que se quiere acceder, S2 la escritura del dato y S3 el bit de stop correspondiente y la preparación para la siguiente transmisión.

4.2.2.3 Prgramdacver2.vhd

Es el encargado de inicializar el RAMDAC, programa el modo de funcionamiento, 8 bits 256 colores escribiendo en los registros adecuados. Carga también la configuración de las LUT´s internas para cada uno de los colores de modo que tengamos una paleta correspondiente a una escala de grises. (256 niveles de gris).

El módulo consta fundamentalmente de un divisor de reloj, 3 contadores para llevar la cuenta de registros escritos y del valor que se programa en las memorias internas del RAMDAC y un controlador y buffers triestado, ya que después de la configuración dejamos los buses en alta impedancia..



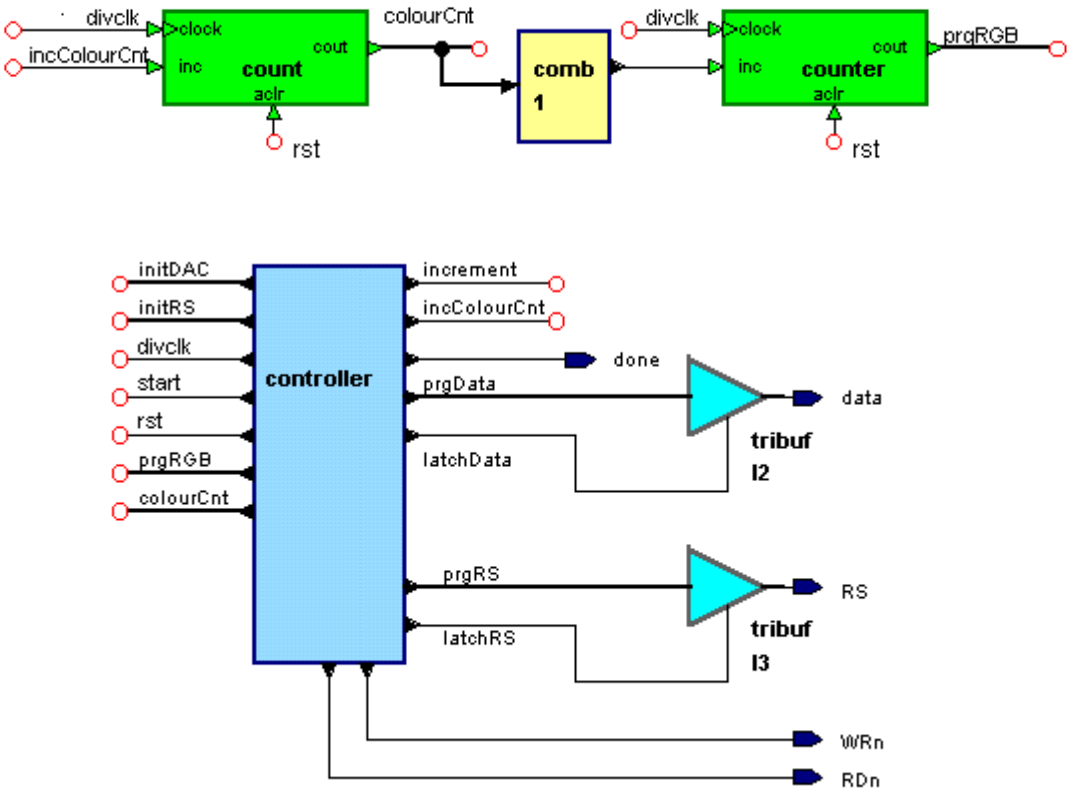


Figura 46. Diagrama RT del módulo prgRamdacver2.vhd

El controlador detecta la señal de start y comienza la configuración, gestiona los incrementos de los diferentes contadores y recoge las palabras adecuadas para configurar el dispositivo, al terminar activa la señal Done y deja los buses en alta impedancia.

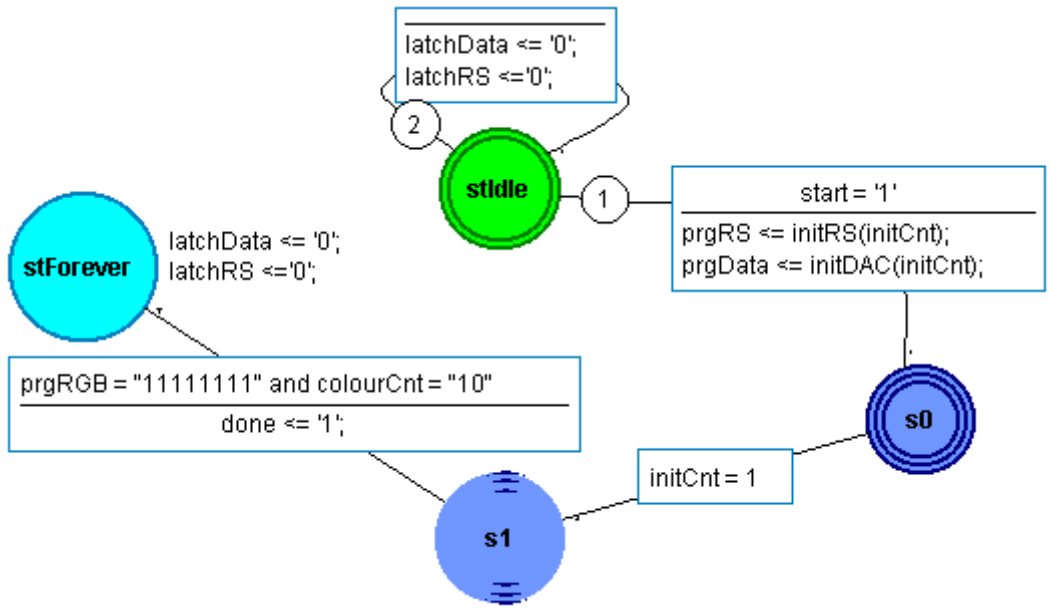


Figura 47. Diagrama de estados del controlador de prgRamdacver2.vhd

En esta representación hemos usado 2 estados jerárquicos S0 y S1 que se desglosan a continuación.

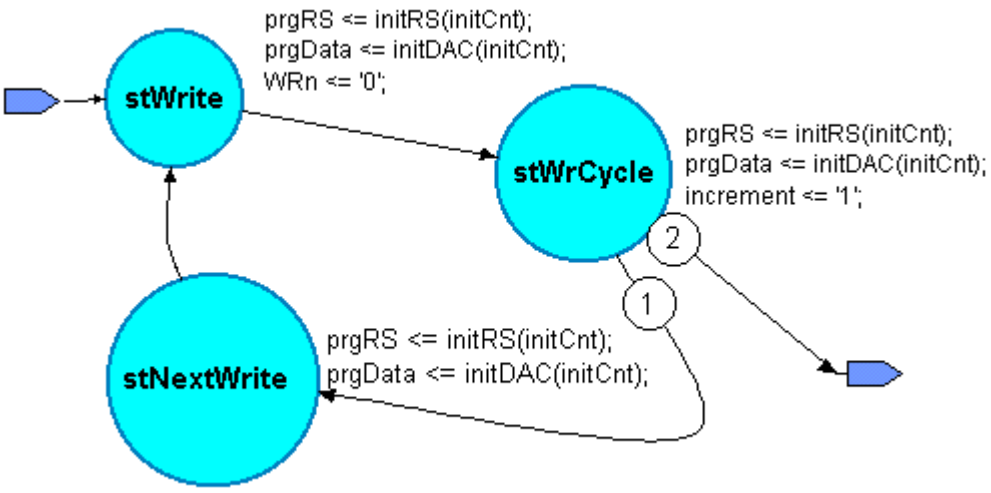


Figura 48. Estados internos a S0.

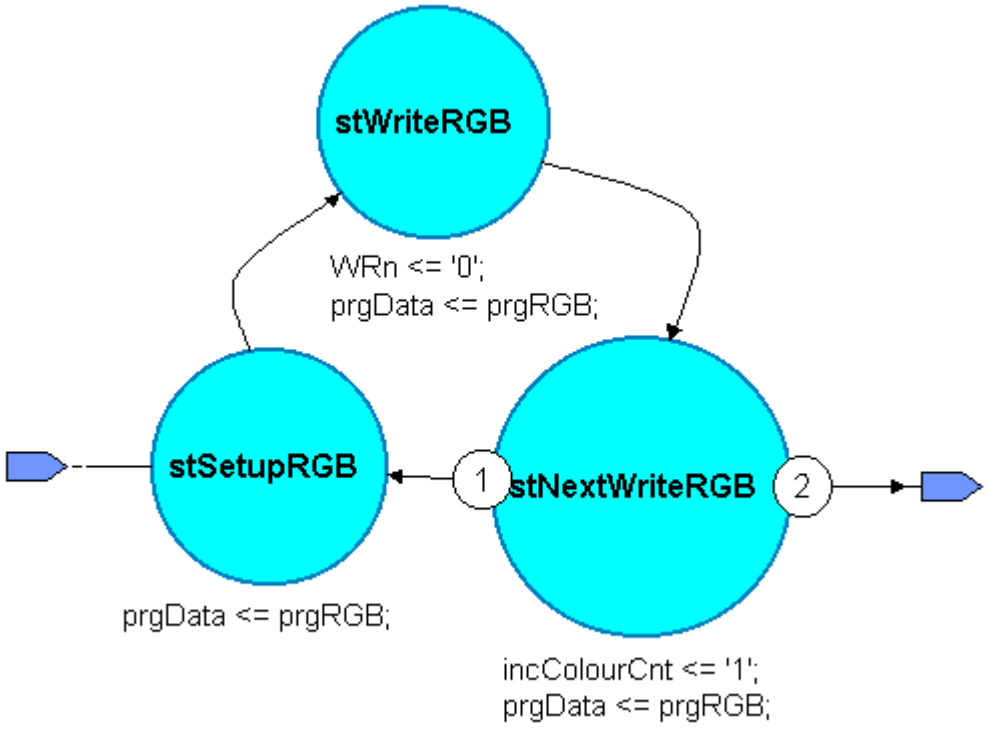


Figura 49. Estados internos a S1.

4.2.2.4 Vgacore.vhd.

El módulo vga genera los pulsos de sincronismo horizontal y vertical necesarios para la correcta temporización de la señal de video resultante, genera además una señal de borrado para el RAMDAC, que pone a nivel de negro la señal en intervalos de sincronismo. Los contadores de píxeles y líneas que contiene sirven al módulo videoprocess.vhd para generar direcciones de lectura a las memorias SRAM.

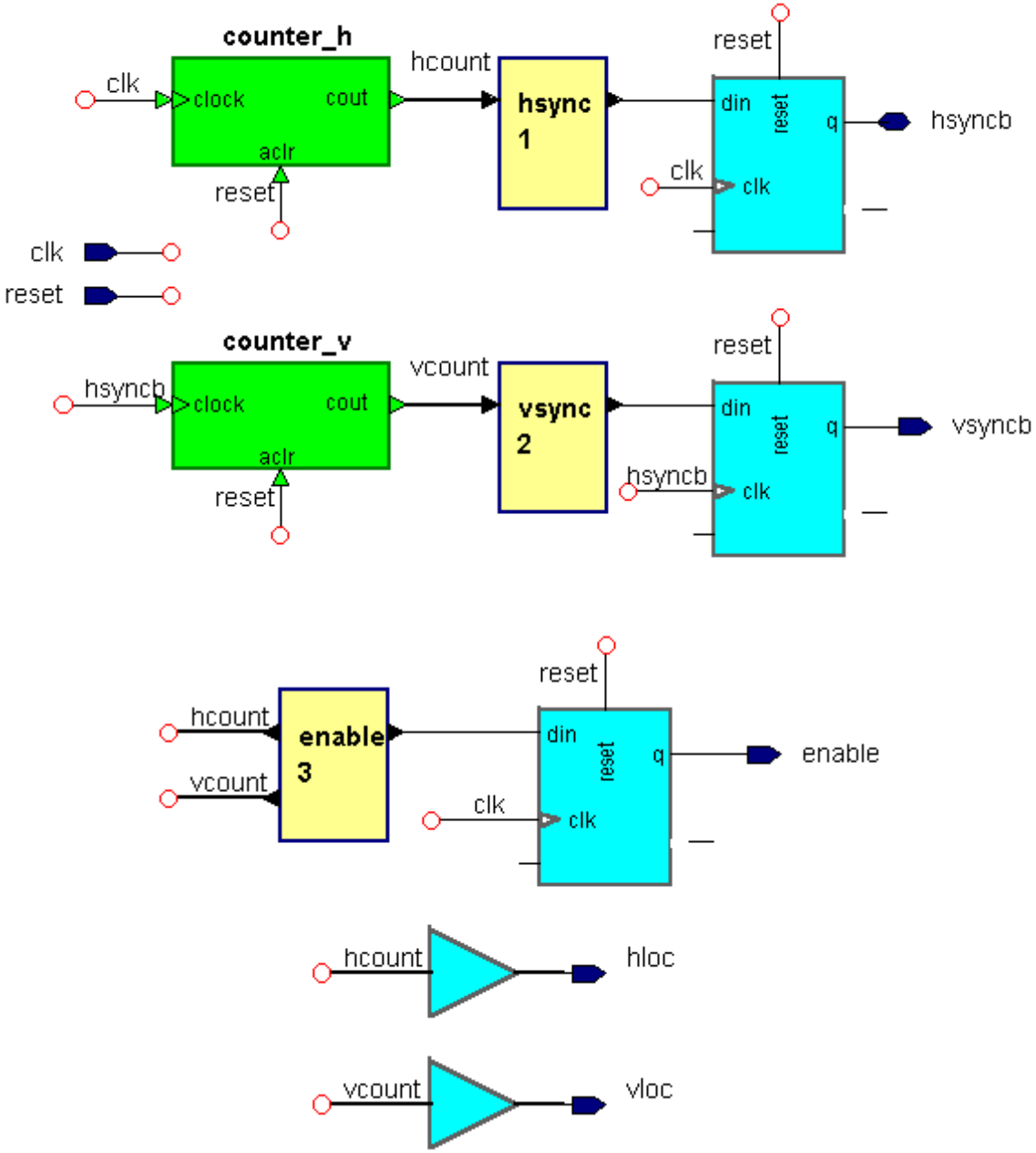


Figura 50. Diagrama RT del módulo vgacore.vhd

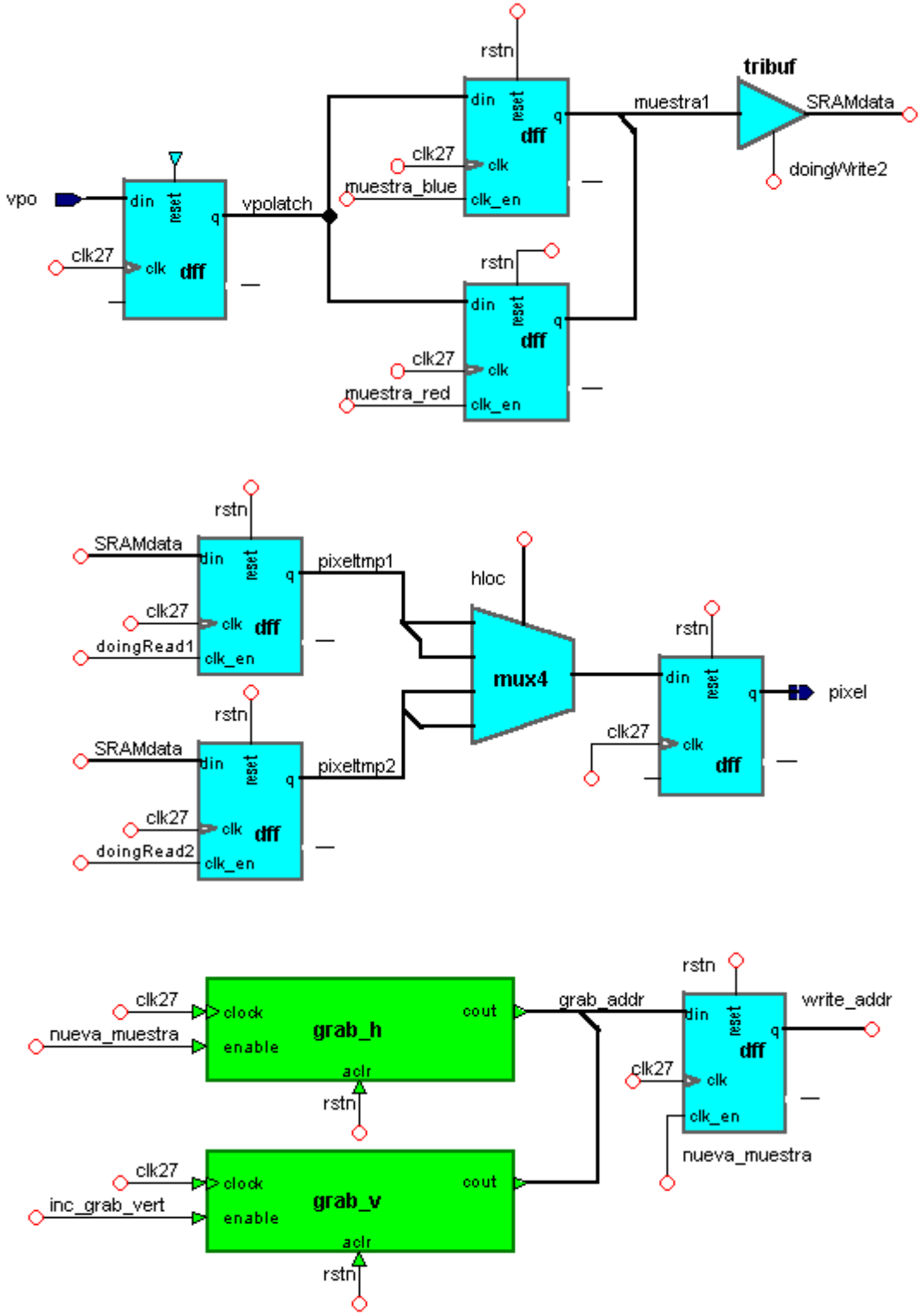
Consta, principalmente de 2 contadores, horizontal para píxeles y vertical para líneas. Unos módulos combinacionales comparadores generan las señales de sincronismo horizontal y vertical respectivamente, estas salidas están registradas. La señal de enable está condicionada por la posición horizontal y vertical y también está registrada. Finalmente los contadores son ofrecidos en puertos de salida. Este módulo no necesita de controlador.

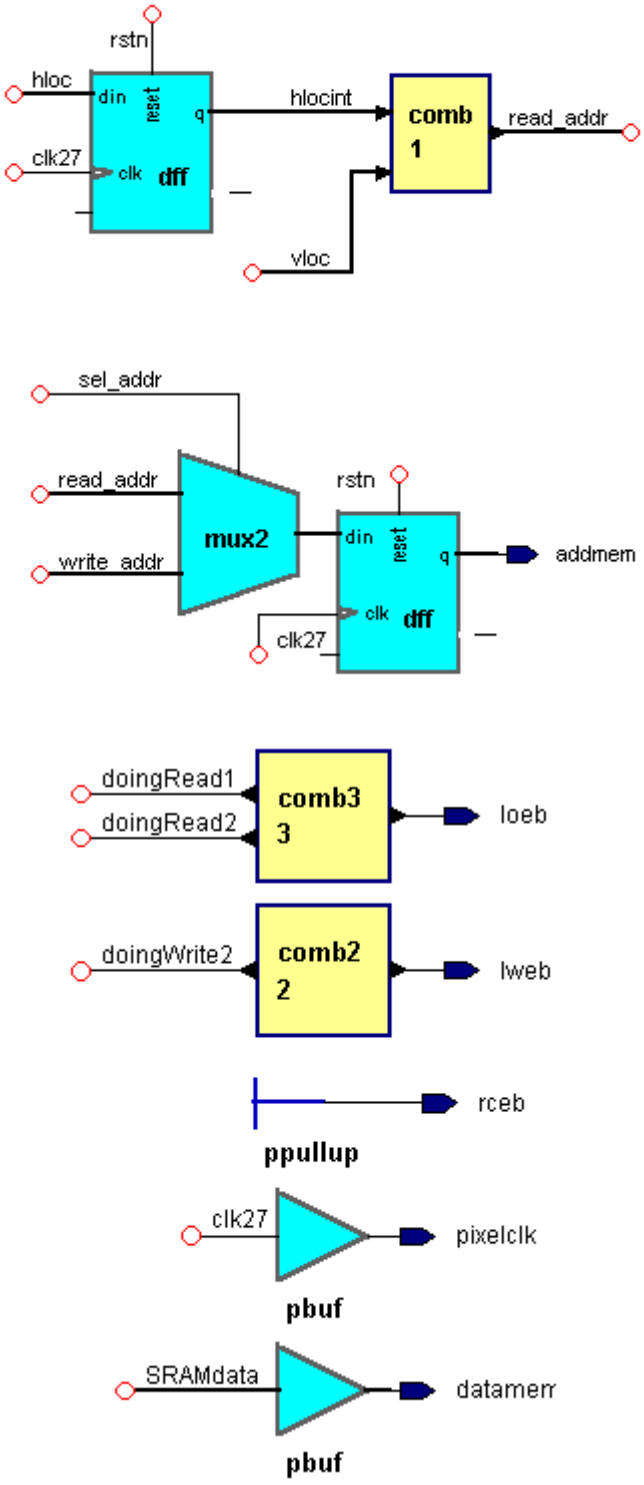
4.2.2.5 Videoprocess.vhd

El módulo videoprocess realiza todas las transformacions requeridas a la señal de vídeo. Decodifica los datos procedentes de las muestras entregadas por el decoder, localiza las referencias de temporización (TRS) de la señal de vídeo y extrae las componentes de luminancia y diferencia de color. Las muestras de luminancia son almacenadas temporalmente en un buffer y formando un bus de 16 bits, agrupando 2 muestras, son escritas en la memoria SRAM. El módulo videoprocess incluye una instancia de vgacore del cúal utiliza las contadores horizontales y verticales para generar las direcciones de lectura de memoria. Las

muestras leídas son también almacenadas temporalmente en un búffer para después ser entregadas al RAMDAC y generar las señales de vídeo de salida.

El módulo videoprocess gestiona tambíen los ciclos de acceso a memoria tanto de lectura como escritura, multiplexándolos en el tiempo y así evitar conflictos en la utilización de los buses.




rstn
hloc
din
reset
q
hlocint
comb
1
read_addr
clk27
clk
dff
vloc
sel_addr
read_addr
mux2
rstn
write_addr
din
reset
q
addmem
clk
dff
clk27
doingRead1
comb3
3
doingRead2
loeb
doingWrite2
comb2
2
lweb
rceb
ppullup
clk27
pixelclk
pbuf
SRAMdata
datamem
pbuf

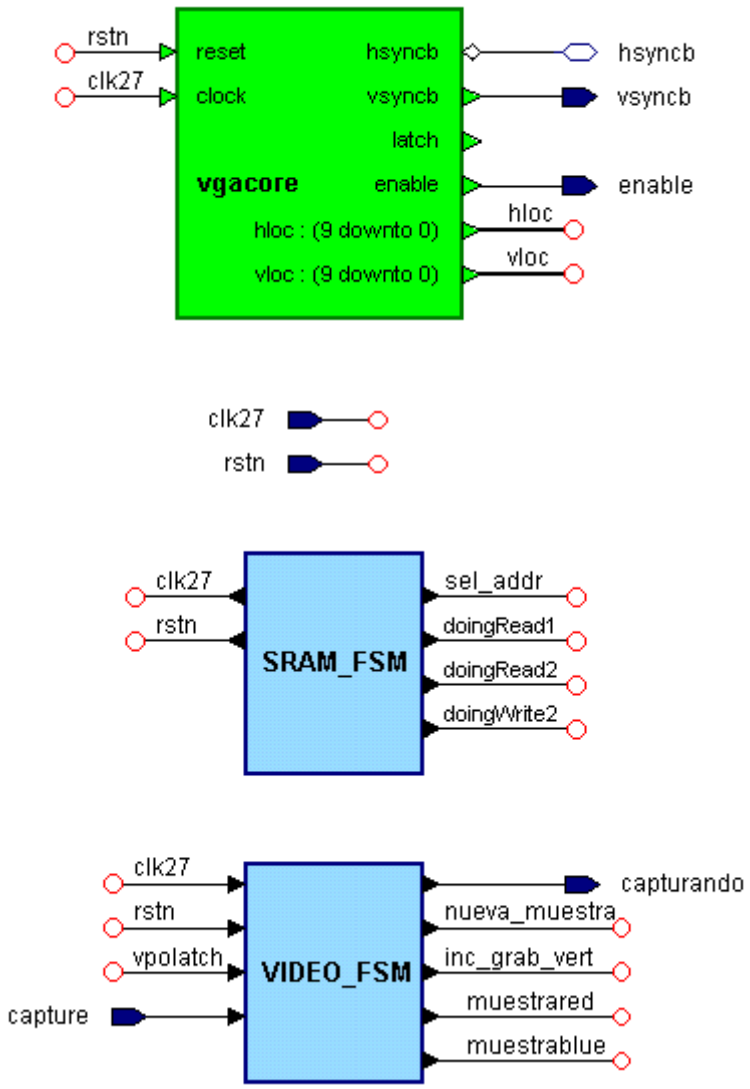


Figura 51. Diagrama RT del módulo videoprocess.vhd

El módulo videoprocess se compone de 2 controladores, uno para accesos a memoria y otro para decodificar video. Junto con el componente vgacore generan todas las señales necesarias para controlar la ruta de datos de la señal de video desde que es capturada hasta que se manda al RAMDAC para su visualización.

A continuación se presentan los diagramas de los controladores anteriormente comentados.

El controlador que decodifica los datos procedentes del decoder no tiene una representación gráfica sencilla como diagrama de estados por lo que los diagramas que se presentan a continuación son sólo una aproximación a la máquina de estados finitos que representa, para más detalles acudir a los listados de código en el apéndice.

En la siguiente figura se muestra la estrategia seguida para tratar convenientememte la señal. Desde el estado inactivo se busca el comienzo de un nuevo frame, toda la información relativa a la temporización de una señal de video viene en los llamados TRS, que es una secuencia de 4 palabras como se ha explicado anteriormente, para ello se desarrolla una especie de subrutina que gestiona que tipo de información contiene contiene el TRS que se trata en cada momento. Esa subrutina podrá ser llamada desde cualquier estado que requiera la localización de un TRS.

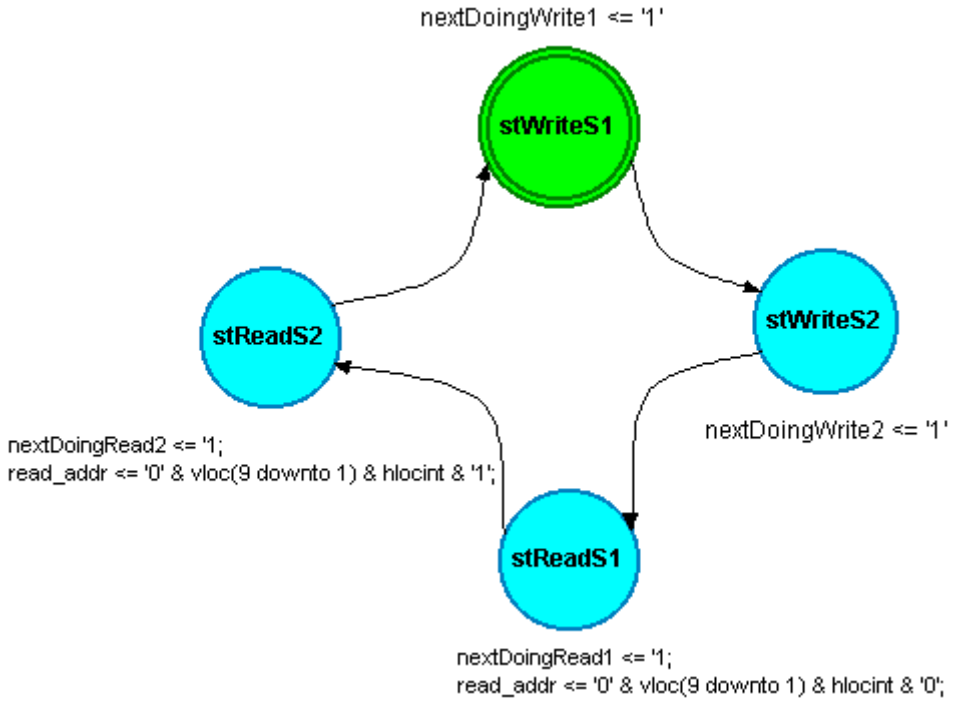


Figura 52. Diagrama de estados del controlador de memoria.

Así, una vez localizado el comienzo de un nuevo frame, se localiza la primera línea activa de vídeo, el estado S0 es un estado jerárquico que procesa las muestras de la señal en sus componentes de luminancia y crominancia. Si se alcanza el final de línea nos disponemos a localizar la siguiente, así hasta lacanzar el final de líneas activas del presente frame. La siguiente operación será, nuevamente localizar un nuevo frame para seguir procesando.

Si en algún momento se detecta alguna incoherencia en las muestras de video se salta al estado de error.

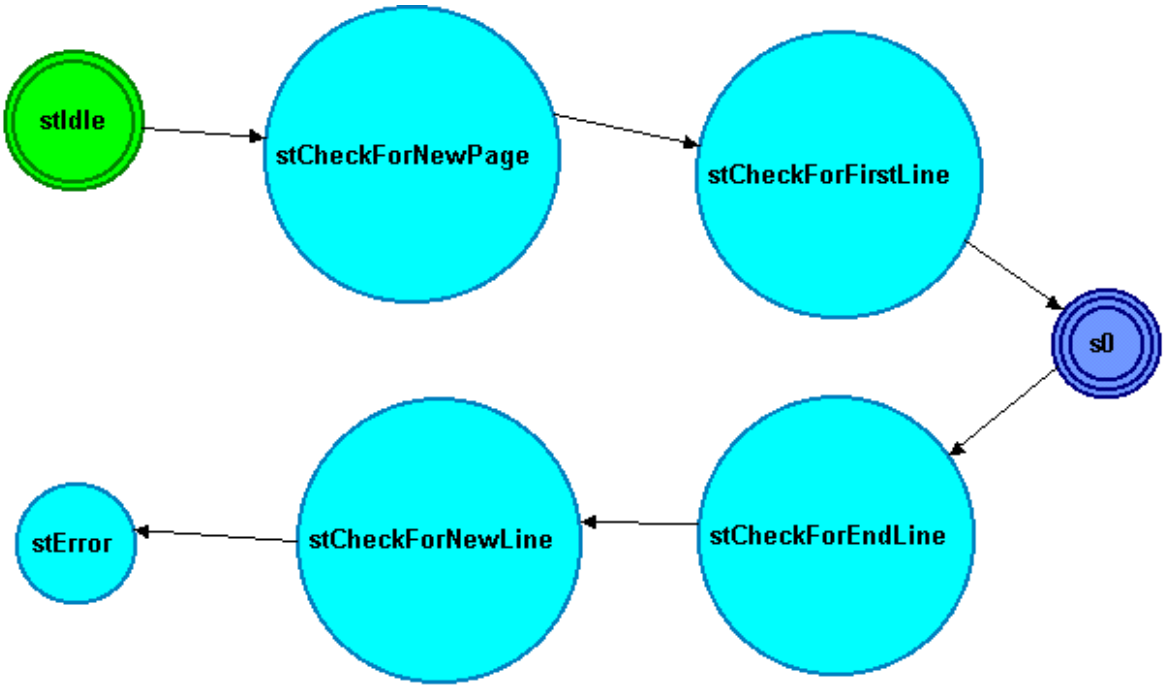


Figura 53. Diagrama principal de estados del controlador de vídeo

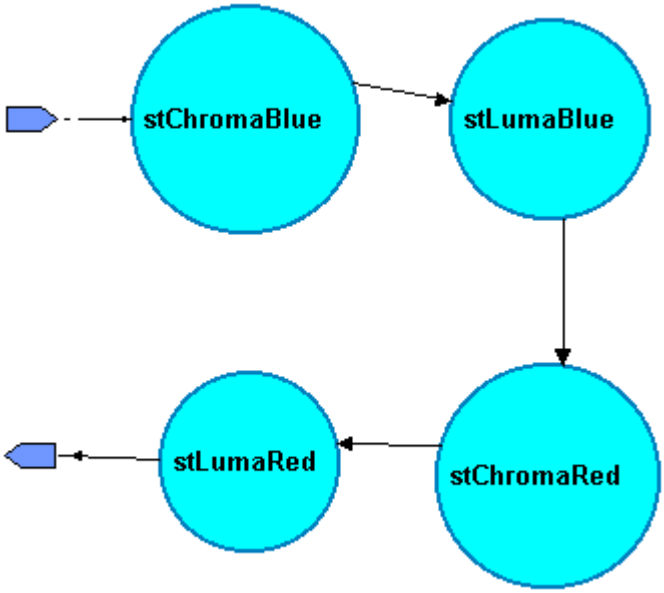


Figura 54. Subdiagrama que procesa las muestras de vídeo.

Son necesarios 4 estados para controlar las diferentes componentes que podemos tener dentro de la trama de video. Sabemos que las componentes vienen multiplexadas con la secuencia B-Y,Y,R-Y,Y,B-Y,Y... sucesivamente, cada 2 muestras de luminancia viene una de cada componente de color.

La subrutina que gestiona los TRS y que puede ser llamada desde cualquier estado del diagrama que requiera localizar algun punto de temporización será la siguiente, solo se dibujan tres estados porque el cuarto corresponderá al retorno desde el diagrama principal presentado anteriormente.

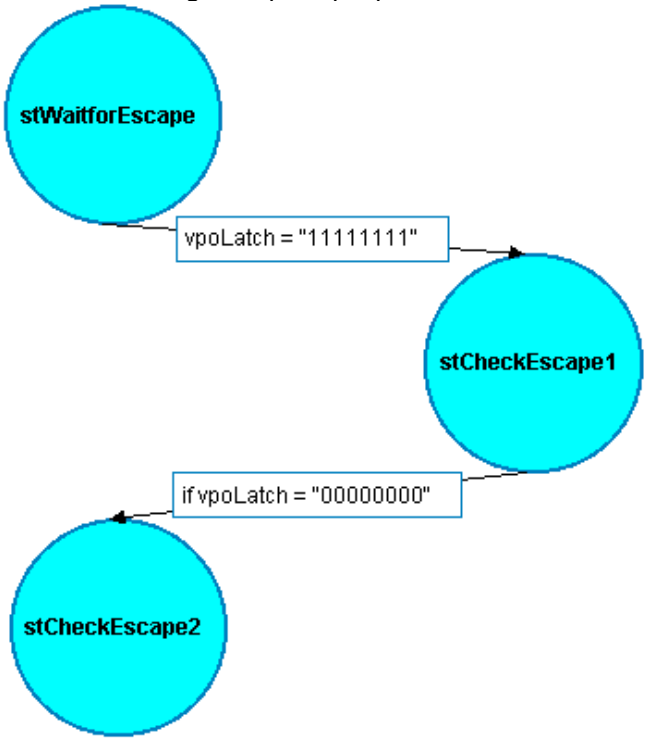


Figura 55. Subrutina que procesa los TRS de la señal de vídeo.

4.2.2.6 SAA713.vhd

Este módulo forma el nivel más alto de jerarquía del sistema. Incluye una instancia de los componentes anteriormente expuestos. Su diagrama de bloques se expuso anteriormente en esta memoria al describir la arquitectura del sistema. Incluye un controlador que gestiona la inicialización de los dispositivos y proporciona los interfaces hacia el exterior del diseño. El diagrama de estados del controlador que incorpora este módulo se puede contemplar en la siguiente figura.

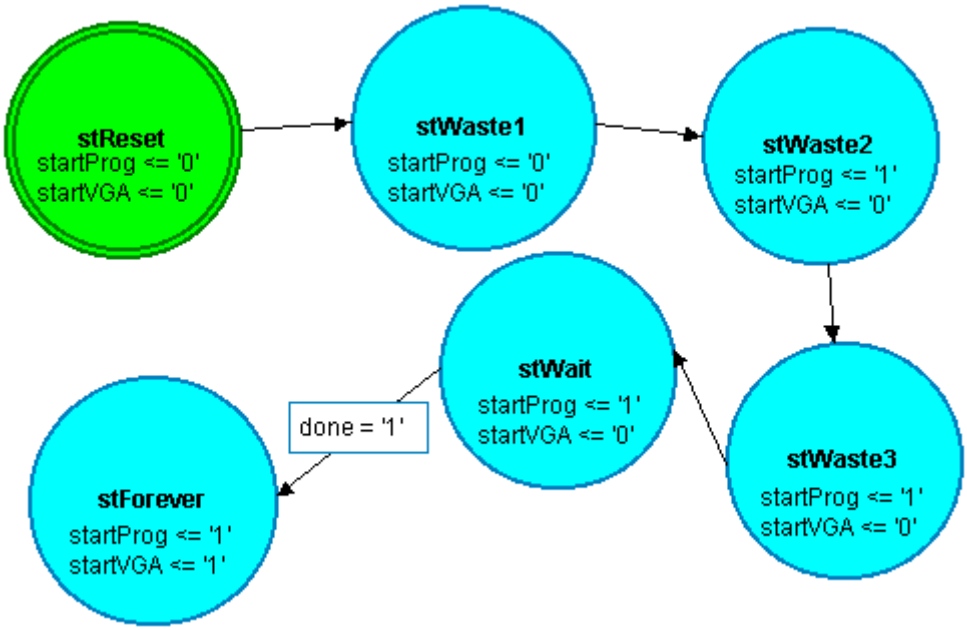


Figura 56. Diagrama de estados del controlador de inicialización.

La señal startProg arranca los controladores de los módulos I2c y prgRamdacver2. Al llegar al estado stWait esperamos a que finalicen ambas inicializaciones, testeando la señal done, para generar la señal startVGA momento en el cúal comienza la captura y visualización de señal de vídeo.

5. CONCLUSIONES

Un dispositivo de lógica programable como una FPGA integra gran cantidad de recursos que permiten la construcción de un sistema digital mas o menos complejo dependiendo de su capacidad. Contando con las herramientas adecuadas y con un lenguaje como VHDL es posible sintetizar circuitos digitales a partir de descripciones de alto nivel. Las prestaciones del circuito obtenido pueden ser mejoradas a nivel de retardos y área utilizando circuitos integrados específicos, la ventaja de una FPGA es que al ser reconfigurable permite la proyección de diferentes diseños dentro del mismo dispositivo.

En este caso se ha sintetizado un sistema que permite el procesado de una señal de vídeo y la configuración de todos los dispositivos externos involucrados en el sistema.

Posibles ampliaciones al proyecto aquí finalizado radican en la consecución de un sistema de procesado de vídeo a color, con posibilidad de escalado de la imagen a diferentes resoluciones. Para ello sería necesario gestionar la memoria de imagen SRAM con diferentes protocolos que permitan superar el ancho de banda de comunicación necesario, así como un módulo conversor de espacio de color de Y,R-Y,B-Y a RGB. La mayor limitación no obstante radica en la frecuencia de reloj máxima que pueda soportar el sistema. Para procesar video en tiempo real se requiere que las operaciones que se realicen utilicen un periodo de reloj lo suficientemente pequeño como para que todas puedan ejecutarse en tiempo adecuado a la captura y posterior visualización, utilización compartida de los buses de acceso a memoria etc...

Al ser la FPGA un dispositivo de propósito general puede verse penalizada en cuanto a retardos en las interconexiones internas frente a un ASIC, sin embargo debido a su flexibilidad pueden utilizarse métodos alternativos como paralelismo o pipelining de las operaciones a ejecutar.

6. BIBLIOGRAFIA Y REFERENCIAS.

SMPTE 125M-1995 -- Component Video Signal 4:2:2 – Bit-Paralell Digital Interface

Rec. UIT-R BT.656-4 Interfaces para las señales de vídeo con componentes digitales en sistemas de televisión de 525 líneas y 625 líneas

Rec. UIT-R BT.470-6 Sistemas de televisión convencional.

Bt481A 110MHz 256-Word Color Palette RAMDAC, Datasheet Brooktree Corporation

SAA7113H 9-bit Video input processor, Datasheet Philips Semiconductor

DS003 (v2.3) Field Programmable Gate Arrays, Xilinx Virtex Datasheet

AS7C4096 512k x 8 CMOS SRAM, Datasheet Alliance Semiconductor

XSV Board v1.0 Manual Xess Corporation

Digital Television Fundamentals: design and installation of video and audio systems.
Michael Robin, Michel Poulin. 2nd Edition
Mc Graw-Hill

Video Demystified: a handbook for the digital engineer.
Keith Jack. 2nd Edition
HighText Publications

7. APÉNDICE

Se incluyen los listados de código VHDL de todos los módulos constituyentes del sistema.

```
----------------------------------------------------------------------------------------------------------------------
--Modulo DEBOUNCER.VHD
--Realiza una eliminacion de reboter en SW2 para realizar congelados de imagen
--Al pulsar congela la imagen, al despulsar sigue con la captura.
--Se basa en la utilizacion de un sincronizador para tratar eventos asincronos
--como la pulsacion del switch y lanza un temporizador hasta su estabilizacion
--al despulsar ejecuta las mismas operaciones, la salida xDeb esta filtrada de
--rebotes respecto de x.
----------------------------------------------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity debouncer is
port(
    rst: in std_logic;
    clk: in std_logic;
    x: in std_logic;
    xDeb: out std_logic;
    xDebFallingEdge: out std_logic;
    xDebRisingEdge: out std_logic
  );
end debouncer;

architecture debouncerArch of debouncer is

signal xSync: std_logic;
signal startTimer, timerEnd: std_logic;

begin

--Forma una estructura de 2 DFF para sincronizar eventos asincronos
--con el reloj del sistema
sincronizador:process(rst, clk)
variable aux1: std_logic;
begin
    if (rst='0') then
      aux1 := '1';
      xSync <= '1';
    elsif (clk'event and clk='1') then
      xSync <= aux1;
      aux1 := x;
    end if;
end process sincronizador;

--Temporizador para eliminacion de rebotes
temporizador:process(rst, clk)
constant timeOut: std_logic_vector (19 downto 0) := "11111000100101101000";
variable count: std_logic_vector (19 downto 0);

begin
    if (count=timeOut) then
      timerEnd <= '1';
    else
      timerEnd <= '0';
    end if;
    if (rst='0') then
      count := timeOut;
    elsif (clk'event and clk='1') then
      if (startTimer='1') then
        count := (others=>'0');
      elsif (timerEnd='0') then
        count := count + 1;
      end if;
    end if;
end process temporizador;

controlador:process(xSync, rst, clk)
type states is (waitingPression, pressionDebouncing, waitingDepression, depressionDebouncing);
variable state: states;

begin
    xDeb <= '1';
```

```
    xDebFallingEdge <= '0';
    xDebRisingEdge <= '0';
    startTimer <= '0';
    case state is

      when waitingPression =>
        if (xSync='0') then
          xDebFallingEdge <= '1';
          startTimer <= '1';
        end if;

              --Se mantiene a '0' hasta que finaliza el timer
      when pressionDebouncing =>
        xDeb <= '0';

      when waitingDepression =>
        xDeb <= '0';
        if (xSync='1') then
          xDebRisingEdge <= '1';
          startTimer <= '1';
        end if;

              --Se mantiene a '1' hasta que finaliza el timer
      when depressionDebouncing =>
        null;

    end case;
    if (rst='0') then
      state := waitingPression;
    elsif (clk'event and clk='1') then
      case state is

        when waitingPression =>
          if (xSync='0') then
            state := pressionDebouncing;
          end if;

        when pressionDebouncing =>
          if (timerEnd='1') then
            state := waitingDepression;
          end if;

        when waitingDepression =>
          if (xSync='1') then
            state := depressionDebouncing;
          end if;

        when depressionDebouncing =>
          if (timerEnd='1') then
            state := waitingPression;
          end if;

      end case;
    end if;
end process controlador;

end debouncerArch;
```

```
------------------------------------------------------------------------------------------------------------------------
--Modulo PRGRAMDACVER2.VHD
--Realiza la configuración del ramdac y carga las LUT con un valor
--correspondiente a su posicion
--Forma una paleta de escala de grises al escribir el mismo valor en RGB
--Se usa modo de acceso LUT en 8 bits a 27 MHz de pixelclk
------------------------------------------------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity prgramdacver2 is
    port (
        clk: in std_logic;
        rstn: in std_logic;
        start: in std_logic;
        done: out std_logic;
        WRn: out std_logic;
        RDn: out std_logic;
        RS: inout std_logic_vector (2 downto 0);
        data: inout std_logic_vector (7 downto 0)
    );
end prgramdacver2;

architecture prgramdacver2_arch of prgramdacver2 is

type STATETYPE is (stIdle, stWrite, stWrCycle, stNextWrite
                        , stSetupRGB, stWriteRGB, stNextWriteRGB,stForever
                        );
signal presState: STATETYPE;
signal nextState: STATETYPE;

--Tipos que implementan 2 memorias ROM, direcciones y datos
type TWODIMARRAYDAC is array (0 to 5) of std_logic_vector (7 downto 0);
type TWODIMARRAYRS is array (0 to 5) of std_logic_vector (2 downto 0);

--Almacena los valores a escribir en cada registro de configuracion
constant initDAC: TWODIMARRAYDAC:=
( -- DAC(76543210)
                        "00000000",
                        "00000000",
                        "11111111",
                        "00000010",
                        "00000010",
                        "00000000"
);

--Selecciona registro a escribir por medio de las lineas RS[2..0]
constant initRS: TWODIMARRAYRS:=
(  -- RS(210)
                        "110",
                        "000",
                        "010",
                        "000",
                        "010",
                        "000"
);


signal divclk: std_logic;
signal divcnt: std_logic_vector (3 downto 0);

signal initCnt: integer range 0 to 5;
signal increment: std_logic;
signal prgRGB: std_logic_vector (7 downto 0);
signal colourCnt: std_logic_vector (1 downto 0);
signal incColourCnt: std_logic;

signal prgData: std_logic_vector (7 downto 0);
signal prgRS: std_logic_vector (2 downto 0);
signal latchData: std_logic;
signal latchRS: std_logic;
```

```
begin

--Contador que divide la frecuencia del reloj de entrada para relajar tiempos
--de acceso al RAMDAC
divisor_clk:process (rstn, clk)
	begin
		if rstn = '0' then
			divcnt <= "0000";
		elsif clk'event and clk = '1' then
			case divcnt is
				when "0000" => divcnt <= "0001";
				when "0001" => divcnt <= "0010";
				when "0010" => divcnt <= "0011";
				when "0011" => divcnt <= "0100";
				when "0100" => divcnt <= "0101";
				when "0101" => divcnt <= "0110";
				when "0110" => divcnt <= "0111";
				when "0111" => divcnt <= "1000";
				when "1000" => divcnt <= "1001";
				when "1001" => divcnt <= "1010";
				when "1010" => divcnt <= "1011";
				when "1011" => divcnt <= "1100";
				when "1100" => divcnt <= "1101";
				when "1101" => divcnt <= "1110";
				when "1110" => divcnt <= "1111";
				when "1111" => divcnt <= "0000";
				when others => divcnt <= "0000";
			end case;
		end if;
end process divisor_clk;

divclk <= divcnt(3);

--No realizamos lecturas de RAMDAC
RDn <= '1';

--Cuentan los registros y valores que se escriben
contadores:process (rstn, divclk)
	begin
		if rstn = '0' then
			presState <= stIdle;
			initCnt <= 0;
			colourCnt <= (others => '0');
			prgRGB <= (others => '0');
		elsif divclk'event and divclk = '1' then
			presState <= nextState;
			if increment = '1' then
				if initCnt < 5 then
					initCnt <= initCnt + 1;
				else
					initCnt <= 0;
				end if;
			end if;
			if incColourCnt = '1' then
				if colourCnt = "10" then
					colourCnt <= "00";
					prgRGB <= prgRGB + 1;
				else
					colourCnt <= colourCnt + 1;
				end if;
			end if;
		end if;
end process contadores;

controlador_RamDac:	process (presState, start, initCnt,prgRGB,colourCnt)
	begin

		WRn <= '1';
		increment <= '0';
		incColourCnt <= '0';
		prgData <= (others => '0');
		prgRS <= "001";
		latchData <= '1';
```

```
                latchRS <= '1';
                done <= '0';

                case presState is
                        when stIdle =>
                                if start = '1' then
                                        nextState <= stWrite;
                                        --Se escriben los valores almacenados en los arrays
                                        prgRS <= initRS(initCnt);
                                        prgData <= initDAC(initCnt);
                                else
                                        nextState <= stIdle;
                                        latchData <= '0';
                                        latchRS <='0';
                                end if;

                        when stWrite =>
                                nextState <= stWrCycle;
                                prgRS <= initRS(initCnt);
                                prgData <= initDAC(initCnt);
                                WRn <= '0';

                        when stWrCycle =>
                                if initCnt = 1 then
                                        nextState <= stSetupRGB;

                                else
                                        nextState <= stNextWrite;
                                end if;
                                prgRS <= initRS(initCnt);
                                prgData <= initDAC(initCnt);
                                increment <= '1';

                        when stNextWrite =>
                                nextState <= stWrite;
                                prgRS <= initRS(initCnt);
                                prgData <= initDAC(initCnt);

                        when stSetupRGB =>
                                nextState <= stWriteRGB;

                        when stWriteRGB =>
                                nextState <= stNextWriteRGB;
                                WRn <= '0';

                        when stNextWriteRGB =>
                                --La ultima posicion y ultimo byte que se escribe en LUT
                                if prgRGB = "11111111" and colourCnt = "10" then
                                        nextState <= stForever;
                                        done <= '1';
                                else
                                        nextState <= stSetupRGB;
                                end if;
                                incColourCnt <= '1';

                        when stForever =>
                                        latchData <= '0';
                                        latchRS <='0';
                                        nextState <= stForever;
                end case;
                --Escribimos datos en LUT
                if presState = stSetupRGB or presState = stWriteRGB or presState = stNextWriteRGB then
                        prgData <= prgRGB;
                end if;
        end process controlador_RamDac;

--Deja las lineas en alta impedancia cuando ha terminado de configurar
data <= prgData when latchData = '1' else (others => 'Z');
RS <= prgRS when latchRS = '1' else (others => 'Z');

end prgramdacver2_arch;
```

```
----------------------------------------------------------------------------------------------------------------------
--Modulo I2c.vhd
--Realiza las rutinas de inicialización del decoder de video SAA7113
--Mantiene las constantes de inicializacion, direcciones y datos a escribir
--en dos ROM indexadas segun el registro a escribir en cada momento.
--Se activa al configurar la FPGA por medio de la senal start, al finalizar
--activa done <= '1'.
----------------------------------------------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity i2c is
port(
                        clk: in std_logic;
                        rst: in std_logic;
                        start: in std_logic;
                        done: out std_logic;
                        scl: out std_logic;
                        sda: out std_logic
        );
end i2c;

architecture i2c_arch of i2c is

--Direccion base de SAA71113
constant SAA7113ADDR: std_logic_vector(7 downto 0):="01001010"; --x4a
--Typo que construye los arrays de inicializacion
type ARRAY_INIT is array (0 to 50) of std_logic_vector (7 downto 0);

--Direcciones de los registros
constant DIR_SAA7113: ARRAY_INIT:=
(
        "00000001",     --x01
        "00000010",     --x02
        "00000011",     --x03
        "00000100",     --x04
        "00000101",     --x05
        "00000110",     --x06
        "00000111",     --x07
        "00001000",     --x08
        "00001001",     --x09
        "00001010",     --x0a
        "00001011",     --x0b
        "00001100",     --x0c
        "00001101",     --x0d
        "00001110",     --x0e
        "00001111",     --x0f
        "00010000",     --x10
        "00010001",     --x11
        "00010010",     --x12
        "00010011",     --x13
        "00010101",     --x15
        "00010110",     --x16
        "00010111",     --x17
        "01000000",     --x40
        "01000001",     --x41
        "01000010",     --x42
        "01000011",     --x43
        "01000100",     --x44
        "01000101",     --x45
        "01000110",     --x46
        "01000111",     --x47
        "01001000",     --x48
        "01001001",     --x49
        "01001010",     --x4a
        "01001011",     --x4b
        "01001100",     --x4c
        "01001101",     --x4d
        "01001110",     --x4e
        "01001111",     --x4f
        "01010000",     --x50
        "01010001",     --x51
```

```
        "01010010",     --x52
        "01010011",     --x53
        "01010100",     --x54
        "01010101",     --x55
        "01010110",     --x56
        "01010111",     --x57
        "01011000",     --x58
        "01011001",     --x59
        "01011010",     --x5a
        "01011011",     --x5b
        "01011110"      --x5e
);

--DAtos a escribir en los registros
constant DATA_SAA7113: ARRAY_INIT:=
(
        "00001000",     --x08
        "11000011",     --xc3
        "00110011",     --x33
        "00000000",     --x00
        "00000000",     --x00
        "11101001",     --xe9
        "00001101",     --x0d
        "00111000",     --x38
        "00000101",     --x05
        "10000000",     --x80
        "01000111",     --x47
        "01000000",     --x40
        "00000000",     --x00
        "00000001",     --x01
        "00100100",     --x24
        "01001000",     --x48
        "00001000",     --x08
        "01110111",     --x77
        "00000000",     --x00
        "00000000",     --x00
        "00000000",     --x00
        "00000000",     --x00
        "00000010",     --x02
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "11111111",     --xff
        "01000000",     --x40
        "01010100",     --x54
        "00000111",     --x07
        "00000011",     --x03
        "00000000"      --x00
);

--Estados del controlador
type STATE_I2C_PRG is(
stIdle,
stBitStart,
stPonSlaveAddr,
stPonSCL1,
stPonSCL0,
```

```
stDesplazaAddr,
stACKSlaveAddr,
stPonSCL1ACKSlaveAddr,
stPonSCL0ACKSlaveAddr,
stPonSubAddr,
stPonSCL1SubAddr,
stPonSCL0SubAddr,
stDesplazaSubAddr,
stACKSubAddr,
stPonSCL1ACKSubAddr,
stPonSCL0ACKSubAddr,
stPonDato,
stPonSCL1Dato,
stPonSCL0Dato,
stDesplazaDato,
stACKDato,
stPonSCL1ACKDato,
stPonSCL0ACKDato,
stBitStop1,
stBitStop,
stBitStopWait1,
stBitStopWait2,
stBitStopWait3,
stBitStopWait4,
stDone
);

signal presStateI2C, nextStateI2C: STATE_I2C_PRG;

signal bitEscrito: integer range 0 to 7;
signal palabraEscrita: integer range 0 to 63;
signal incBit, incPalabra: std_logic;
signal txDatoReg: std_logic_vector(7 downto 0);
signal loadAddr,loadSubAddr,loadDato: std_logic;

signal datoSerie :std_logic;
signal shift:std_logic;
signal clk_div: std_logic;
signal estado:std_logic_vector(12 downto 0);

begin

--Dividimos el reloj de entrada
divide_clk:process(clk,rst,clk_div)
begin
  if(rst='0') then
            estado <=(others => '0');
            clk_div <= '0';
  elsif ( clk'event and clk ='1' ) then
    if (estado="1000000000000") then
                        estado <=(others => '0');
                        clk_div <= not clk_div;
    else
      estado <= estado + 1;
    end if;
  end if;
end process divide_clk;

--Contadores de bit y registros escritos
contadores:process(clk_div, rst)
begin

            if rst='0' then
                        presStateI2C <= stIdle;
                        bitEscrito <= 0;
                        palabraEscrita <= 0;
            elsif clk_div'event and clk_div='1' then
                        presStateI2C <= nextStateI2C;
                        if incBit='1' then
                                    if bitEscrito = 7 then
                                                bitEscrito <= 0;
                                    else
                                                bitEscrito <= bitEscrito + 1;
                                    end if;
                        end if;
                        if incPalabra='1' then
```

```
                        if palabraEscrita = 50 then
                                palabraEscrita <= 0;
                        else
                                palabraEscrita <= palabraEscrita + 1;
                        end if;
                end if;
        end if;
end process contadores;

--Implementa un registro de desplazamiento cargado con carga paralelo
registro_desplazamiento:process(rst,clk_div)
begin

        if rst='0' then
                txDatoReg <= (others =>'0');
        elsif clk_div'event and clk_div='1' then
                if loadAddr='1' then
                        txDatoReg <= SAA7113ADDR;
                elsif loadSubAddr = '1' then
                        txDatoReg <= DIR_SAA7113(palabraEscrita);
                elsif loadDato='1' then
                        txDatoReg <= DATA_SAA7113(palabraEscrita);
                elsif shift='1' then
                        txDatoReg <= txDatoReg(6 downto 0) & '0';
                end if;
        end if;
        datoSerie <= txdatoReg(7);
end process registro_desplazamiento;


controlador_i2c:process(start,presStateI2C,bitEscrito,palabraEscrita,datoserie)
begin

        incBit <= '0';
        incPalabra <= '0';
        done <= '0';
        loadAddr <= '0';
        loadSubAddr <= '0';
        loadDato <= '0';
        shift <= '0';

        nextStateI2C <= presStateI2C;

        case presStateI2C is

                when stIdle =>
                        if start='1' then
                                nextStateI2C <= stBitStart;
                        end if;
                        scl <= '1';
                        sda <= '1';

                when stBitStart =>
                        sda <= '0';
                        scl <= '1';
                        loadAddr <= '1';
                        nextStateI2C <= stPonSlaveAddr;

                when stPonSlaveAddr =>
                        scl <= '0';
                        sda <= datoserie;
                        nextStateI2C <= stPonSCL1;

                when stPonSCL1 =>
                        scl <= '1';
                        sda <= datoserie;
                        nextStateI2C <= stPonSCL0;

                when stPonSCL0 =>
                        scl <= '0';
                        sda <= datoserie;
                        incBit <= '1';
                        shift <= '1';
                        if bitEscrito = 7 then
                                nextStateI2C <= stACKSlaveAddr;
                        else
```

```
                    nextStateI2C <= stDesplazaAddr;
            end if;

when stDesplazaAddr =>
            scl <= '0';
            sda <= datoserie;
            nextStateI2C <= stPonSCL1;

when stACKSlaveAddr =>
            scl <= '0';
            sda <= '0';
            nextStateI2C <= stPonSCL1ACKSlaveAddr;

when stPonSCL1ACKSlaveAddr =>
            sda <= '0';
            scl <= '1';
            nextStateI2C <= stPonSCL0ACKSlaveAddr;

when stPonSCL0ACKSlaveAddr =>
            sda <= '0';
            scl <= '0';
            loadSubAddr <= '1';
            nextStateI2C <= stPonSubAddr;

when stPonSubAddr =>
            scl <= '0';
            sda <= datoserie;
            nextStateI2C <= stPonSCL1SubAddr;

when stPonSCL1SubAddr =>
            sda <= datoserie;
            scl <= '1';
            nextStateI2C <= stPonSCL0SubAddr;

when stPonSCL0SubAddr =>
            scl <= '0';
            sda <= datoserie;
            incBit <= '1';
            shift <= '1';
            if bitEscrito = 7 then
                        nextStateI2C <= stACKSubAddr;
            else
                        nextStateI2C <= stDesplazaSubAddr;
            end if;

when stDesplazaSubAddr =>
            scl <= '0';
            sda <= datoserie;
            nextStateI2C <= stPonSCL1SubAddr;

when stACKSubAddr =>
            sda <= '0';
            scl <= '0';
            nextStateI2C <= stPonSCL1ACKSubAddr;

when stPonSCL1ACKSubAddr =>
            sda <= '0';
            scl <= '1';
            nextStateI2C <= stPonSCL0ACKSubAddr;

when stPonSCL0ACKSubAddr =>
            sda <= '0';
            scl <= '0';
            loadDato <= '1';
            nextStateI2C <= stPonDato;

when stPonDato =>
            scl <= '0';
            sda <= datoserie;
            nextStateI2C <= stPonSCL1Dato;

when stPonSCL1Dato =>
            sda <= datoserie;
            scl <= '1';
            nextStateI2C <= stPonSCL0Dato;
```

```
                when stPonSCL0Dato =>
                        scl <= '0';
                        sda <= datoserie;
                        incBit <= '1';
                        shift <= '1';
                        if bitEscrito = 7 then
                                nextStateI2C <= stACKDato;
                        else
                                nextStateI2C <= stDesplazaDato;
                        end if;

                when stDesplazaDato =>
                        scl <= '0';
                        sda <= datoserie;
                        nextStateI2C <= stPonSCL1Dato;

                when stACKDato =>
                        sda <= '0';
                        scl <= '0';
                        nextStateI2C <= stPonSCL1ACKDato;

                when stPonSCL1ACKDato =>
                        sda <= '0';
                        scl <= '1';
                        nextStateI2C <= stPonSCL0ACKDato;

                when stPonSCL0ACKDato =>
                        sda <= '0';
                        scl <= '0';
                        nextStateI2C <= stBitStop1;

                when stBitStop1 =>
                        sda <= '0';
                        scl <= '1';
                        nextStateI2C <= stBitStop;

                when stBitStop =>
                        scl <= '1';
                        sda <= '1';
                        incPalabra <= '1';
                        nextStateI2C <= stBitStopWait1;

                when stBitStopWait1 =>
                        scl <= '1';
                        sda <= '1';
                        nextStateI2C <= stBitStopWait2;

                when stBitStopWait2 =>
                        scl <= '1';
                        sda <= '1';
                        nextStateI2C <= stBitStopWait3;

                when stBitStopWait3 =>
                        scl <= '1';
                        sda <= '1';
                        nextStateI2C <= stBitStopWait4;

                when stBitStopWait4 =>
                        scl <= '1';
                        sda <= '1';
                        if palabraEscrita = 50 then
                                nextStateI2C <= stDone;
                        else
                                nextStateI2C <= stBitStart;
                        end if;

                when stDone =>
                        scl <= '1';
                        sda <= '1';
                        done <= '1';
                        nextStatei2c <= stDone;
        end case;
end process controlador_i2c;

end i2c_arch;
```

```
----------------------------------------------------------------------------------------------------------------------
--Modulo VGACORE.VHD
--Genera la temporizacion de refresco para presentacion de imagen en monitor
--Genera las señales de sincronismo horizontales y verticales
--Implementa un contador de pixeles y lineas que permiten indexar las
--SRAM para obtener el pixel adecuado a presentar en pantalla.
--El formato de imagen es de 720pixels x 576lineas progresivo.
--La frecuencia de refresco vertical son 50 Hz.
--La senal enable fuerza las salidas del RAMDAC a nivel de negro
--hloc y vloc sirven para indexar las memorias SRAM, esas senales seran
--utilizadas en el modulo videoprocess
----------------------------------------------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity vgacore is
	generic (
		H_SIZE : integer := 720;
		V_SIZE : integer := 576
	);
	port
	(
		reset: in std_logic;
		clock: in std_logic;
		hsyncb: buffer std_logic;
		vsyncb: out std_logic;
		latch: out STD_LOGIC;
		enable: out STD_LOGIC;
		hloc: out std_logic_vector(9 downto 0);
		vloc: out std_logic_vector(9 downto 0)
	);
end vgacore;

architecture vgacore_arch of vgacore is

--Declaracion de constantes para temporizacion de pulsos horizontales
constant H_PIXELS: integer:= H_SIZE;
constant H_FRONTPORCH:  integer:= 42;
constant H_SYNCTIME: integer:= 70;
constant H_BACKPORCH: integer:= 31;
constant H_PERIOD: integer:= H_SYNCTIME + H_PIXELS + H_FRONTPORCH + H_BACKPORCH;

--Declaracion de constantes para temporizacion de pulsos verticales
constant V_LINES: integer:= V_SIZE;
constant V_FRONTPORCH: integer:= 27;
constant V_SYNCTIME: integer:= 6;
constant V_BACKPORCH: integer:= 15;
constant V_PERIOD: integer:= V_SYNCTIME + V_LINES + V_FRONTPORCH + V_BACKPORCH;

signal hcnt: std_logic_vector(10 downto 0);	--Contador de pixeles
signal vcnt: std_logic_vector(9 downto 0);	--Contador de lineas

begin

--El contador de pixeles se actualiza a eventos de reloj
contador_horizontal: process(clock, reset)
begin

	if reset = '0' then
		hcnt <= (others => '0');
	elsif (clock'event and clock = '1') then
		if hcnt < H_PERIOD then
			hcnt <= hcnt + 1;
		else
			hcnt <= (others => '0');
		end if;
	end if;
end process contador_horizontal;

--El contador de lineas se actualiza a eventos de sincronismos horizontales
contador_vertical: process(hsyncb, reset)
begin

	if reset='0' then
```

```
                vcnt <= (others => '0');
        elsif (hsyncb'event and hsyncb = '1') then
                if vcnt < V_PERIOD then
                        vcnt <= vcnt + 1;
                else
                        vcnt <= (others => '0');
                end if;
        end if;
end process contador_vertical;

--Genera pulsos de sincronismo horizontal dependiendo del valor del contador
--horizontal
sincro_horizontal: process(clock, reset)
begin

        if reset = '0' then
                hsyncb <= '1';
        elsif (clock'event and clock = '1') then
                if (hcnt >= (H_FRONTPORCH + H_PIXELS) and hcnt < (H_PIXELS + H_FRONTPORCH + H_SYNCTIME)) then
                        hsyncb <= '0';
                else
                        hsyncb <= '1';
                end if;
        end if;
end process sincro_horizontal;

--Los pulsos de sincronismo vertical dependen del contador de lineas
sincro_vertical: process(hsyncb, reset)
begin

        if reset = '0' then
                vsyncb <= '1';
        elsif (hsyncb'event and hsyncb = '1') then
                if (vcnt >= (V_LINES + V_FRONTPORCH) and vcnt < (V_LINES + V_FRONTPORCH + V_SYNCTIME)) then
                        vsyncb <= '0';
                else
                        vsyncb <= '1';
                end if;
        end if;
end process sincro_vertical;

latch <= not reset;

--Existen zonas de borrado tanto en sincronismo horizontal como vertical
borrado: process (clock)
begin
        if clock'event and clock = '1' then
                if hcnt >= H_PIXELS or hcnt <= 15 or vcnt >= V_LINES then
                        enable <= '0';
                 else
                        enable <= '1';
                 end if;
        end if;
end process borrado;

hloc <= hcnt(9 downto 0);
vloc <= vcnt(9 downto 0);

end vgacore_arch;
```

```
----------------------------------------------------------------------------------------------------------------
--Modulo VIDEOPROCESS.VHD
--Se realiza la decodificacion de video digital ITU-R BT.656 entregada por
--el decoder. Se extraen las referencias de temporizacion TRS y se
--localizan los EAV y SAV embebidos en la senal de video.
--Las muestras correspondientes a luminancia Y son escritas en SRAM.
--Las muestras se empiezan a procesar cuando se detecta el comienzo de un
--frame. Los accesos a memoria son de 16 bits, como la codificacion de video
--se realiza en 8 bits en cada acceso a memoria se escriben 2 pixeles.
--Este modulo funciona con un reloj de 27 MHz procedente del decoder.
--Se realizan 2 lecturas consecutivas de memoria y se almacenan en un buffer
--temporal hasta que son solicitadas por los contadores del modulo VGACORE
----------------------------------------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity videoprocess is
port(
        clk27: in std_logic;                            --Reloj del decoder
        vpo: in std_logic_vector(7 downto 0);           --Datos de video
        rstn: in std_logic;
        capture: in std_logic;                          --Senal que habilita la captura
        pixel: out std_logic_vector(7 downto 0);        --Pixel hacia RAMDAC
        pixelclk: out std_logic;                        --Reloj de rAMDAC 27MHz
        hsync: buffer std_logic;
        vsync: out std_logic;
        enable: out std_logic;
        lceb, loeb, lweb: out std_logic;                --Control de memorias
        addmem: out std_logic_vector(18 downto 0);
        datamem: inout std_logic_vector(15 downto 0);
        capturando: out std_logic                       --Senal de estado
);
end videoprocess;

architecture videoproc_arch of videoprocess is

--Utilizacion del componente VGACORE
component vgacore
        generic (
                H_SIZE : integer := 720;
                V_SIZE : integer := 576
        );
        port
        (
                reset: in std_logic;
                clock: in std_logic;
                hsyncb: buffer std_logic;
                vsyncb: out std_logic;
                latch: out STD_LOGIC;
                enable: out STD_LOGIC;
                hloc: out std_logic_vector(9 downto 0);
                vloc: out std_logic_vector(9 downto 0)
        );
end component;

--Maquina de estados para decodificacion de video digital, extraccion de
--muestras de video y referencias de temporizacion
type VIDEO_STATE_TYPE is (
        stIdle,
        stWaitForEscape,
        stCheckEscape1,
        stCheckEscape2,
        stCheckForNewPage,
        stCheckForFirstLine,
        stChromaBlue,
        stLumaBlue,
        stChromaRed,
        stLumaRed,
        stCheckForEndLine,
        stCheckForNewLine,
        stError
);
```

```
signal presState: VIDEO_STATE_TYPE;
signal nextState: VIDEO_STATE_TYPE;
signal returnState: VIDEO_STATE_TYPE;
signal nextReturnState: VIDEO_STATE_TYPE;

--Maquina de estados que gestiona los accesos a memoria, utiliza dos ciclos
--para hacer una escritura y un ciclo para lectura. Realiza una escritura y
--2 lecturas por pasada para permitir un procesado en tiempo real de la
--señal de video
type SRAM_WRITE_TYPE is (
        stWriteS1,
        stWriteS2,
        stReadS1,
        stReadS2
);

signal presStateRAM: SRAM_WRITE_TYPE;
signal nextStateRAM: SRAM_WRITE_TYPE;

signal SRAMData: std_logic_vector(15 downto 0);
signal OEn, WEn: std_logic;

signal addmemint: std_logic_vector(18 downto 0);

signal field : std_logic;
signal nextField : STD_LOGIC;
signal vpoLatch: std_logic_vector(7 downto 0);

signal grab_addr: std_logic_vector(18 downto 0);
signal grab_cntr_hori: std_logic_vector(9 downto 0);
signal grab_cntr_vert: std_logic_vector(9 downto 0);
signal clr_grab_cntr: std_logic;
signal inc_grab_hori: std_logic;
signal inc_grab_vert: std_logic;

signal doingWrite1, doingWrite2: std_logic;
signal doingRead1, doingRead2: std_logic;
signal nextdoingWrite1, nextdoingWrite2: std_logic;
signal nextdoingRead1, nextdoingRead2: std_logic;

signal nueva_muestra: std_logic;
signal muestra1: std_logic_vector(15 downto 0);
signal muestrablue, muestrared: std_logic;
signal pixeltmp1, pixeltmp2: std_logic_vector(15 downto 0);

signal hlocint: std_logic_vector(7 downto 0);
signal hloc: std_logic_vector(9 downto 0);
signal vloc: std_logic_vector(9 downto 0);

signal error: std_logic;

signal latch: std_logic;

signal read_addr:std_logic_vector(18 downto 0);
signal write_addr: std_logic_vector(18 downto 0);
signal pixelint: std_logic_vector(7 downto 0);

begin

--Instanciamos componente VGACORE
vga_gen: vgacore
        generic map(
                H_SIZE => 720,
                V_SIZE => 576
        )
        port map
        (
                reset => rstn,
                clock => clk27,
                hsyncb => hsync,
                vsyncb => vsync,
                latch => latch,
                enable => enable,
                hloc => hloc,
                vloc => vloc
```

```
	);

--Conexiones
pixelclk <= clk27;
datamem <= SRAMData;
lceb <= '0';
loeb <= OEn;
lweb <= WEn;

--Multiplex que controla el bus de direcciones de memoria para accesos de
--lectura y escritura en que las direccones pueden ser diferentes
addmemint <= write_addr when (nextdoingWrite1='1' or nextdoingWrite2='1') else read_addr;

--Puerto de direcciones registrado
registro_dir:process(clk27, rstn)
begin
	if rstn='0' then
		addmem <= (others =>'0');
	elsif clk27'event and clk27='1' then
		addmem <= addmemint;
	end if;
end process registro_dir;

--Actualizacion de senales de FSM decodificador de video y latcheo del
--video de entrada
decoder_video_clk:process(clk27,rstn)
begin
	if rstn = '0' then
		presState <= stIdle;
		returnState <= stIdle;
		field <= '0';
		vpoLatch <= (others => '0');
	elsif clk27'event and clk27='1' then
		vpoLatch <= vpo;
		presState <= nextState;
		returnState <= nextReturnState;
		field <= nextField;
	end if;
end process decoder_video_clk;

--FSM que recibe los datos de video y realiza la correspondiente decodificacion
--realizando un aextraccion de muestras de luminancia y localizando las
--posiciones del frame dependiendo de los TRS.
--Al localizar el comienzo de un nuevo frame, resetea las direcciones
--de escritura.
decoder_video_fsm:process (presState, vpoLatch, returnState, field, capture)
begin

	clr_grab_cntr <= '0';
	inc_grab_vert <= '0';
	nextReturnState <= returnState;
	nextField <= field;
	error <= '0';
	nueva_muestra <= '0';
	muestrablue <= '0';
	muestrared <= '0';
	capturando <= '1';
	case presState is

		when stIdle =>
			if capture = '1' then
				nextState <= stWaitForEscape;
				nextReturnState <= stCheckForNewPage;
			else
				nextState <= stIdle;
				capturando <= '0';
			end if;

		when stWaitForEscape =>
			if vpoLatch = "11111111" then
				nextState <= stCheckEscape1;
			else
				nextState <= stWaitForEscape;
			end if;

		when stCheckEscape1 =>
```

```
		if vpoLatch = "00000000" then
			nextState <= stCheckEscape2;
		else
			nextState <= stError;
		end if;

	when stCheckEscape2 =>
		if vpoLatch = "00000000" then
			nextState <= returnState;
		else
			nextState <= stError;
		end if;

	when stCheckForNewPage =>
		if vpoLatch(6 downto 5) = "01" then
			nextState <= stWaitForEscape;
			nextReturnState <= stCheckForFirstLine;
			clr_grab_cntr <= '1';
		else
			nextState <= stWaitForEscape;
			nextReturnState <= stCheckForNewPage;
		end if;

	when stCheckForFirstLine =>
		if vpoLatch(6 downto 4) = "000"		then
			nextState <= stChromaBlue;
			nextField <= '0';
		else
			nextState <= stWaitForEscape;
			nextReturnState <= stCheckForFirstLine;
		end if;

	when stChromaBlue =>
		if vpoLatch = "11111111" then
			nextState <= stCheckEscape1;
			nextReturnState <= stCheckForEndLine;
		elsif vpoLatch = "00000000" then
			nextState <= stError;
		else
			nextState <= stLumaBlue;
		end if;

	when stLumaBlue =>
		if vpoLatch /= "11111111" and vpoLatch /= "00000000" then
			nextState <= stChromaRed;
			if field='0' then
				muestrablue <= '1';
				nueva_muestra <= '1';
			end if;
		else
			nextState <= stError;
		end if;

	when stChromaRed =>
		if vpoLatch /= "11111111" and vpoLatch /= "00000000" then
			nextState <= stLumaRed;
		else
			nextState <= stError;
		end if;

	when stLumaRed =>
		if vpoLatch /= "11111111" and vpoLatch /= "00000000" then
			nextState <= stChromaBlue;
			if field='0' then
				nueva_muestra <= '1';
				muestrared <= '1';
			end if;
		else
			nextState <= stError;
		end if;

	when stCheckForEndLine =>
		if vpoLatch(6 downto 4) = "111" then
			nextState <= stIdle;
		elsif vpoLatch(6 downto 4) = "011" then
			clr_grab_cntr <= '1';
```

```
                                        nextState <= stWaitForEscape;
                                        nextReturnState <= stCheckForNewLine;
                                elsif vpoLatch(5 downto 4) = "01" then
                                        inc_grab_vert <= '1';
                                        nextState <= stWaitForEscape;
                                        nextReturnState <= stCheckForNewLine;
                                else

                                nextState <= stError;
                                end if;

                        when stCheckForNewLine =>
                                if vpoLatch(5 downto 4) = "00" then
                                        nextState <= stChromaBlue;
                                        nextField <= vpoLatch(6);
                                else
                                        nextState <= stWaitForEscape;
                                        nextReturnState <= stCheckForNewLine;
                                end if;

                        when stError =>
                                if capture = '1' then
                                        nextState <= stWaitForEscape;
                                        nextReturnState <= stCheckForNewPage;
                                else
                                        nextState <= stError;
                                        error <= '1';
                                end if;
                end case;
end process decoder_video_fsm;

--Actualiza las direcciones de escritura segun se detectan muestras de
--luminancia en la trama de video. Se crea un bus de 16 bits a partir de
--2 muestras consecutivas
control_escrituras:process(clk27, rstn)
begin
        if rstn = '0' then
                muestra1 <= (others => '0');
                grab_cntr_hori <= (others => '0');
                grab_cntr_vert <= (others => '0');
                write_addr <= (others => '0');
        elsif clk27'event and clk27 = '1' then
                if (nueva_muestra = '1' and muestrablue = '1') then
                        muestra1(15 downto 8) <= vpoLatch;
                end if;
                if (nueva_muestra = '1' and muestrared = '1') then
                        muestra1(7 downto 0) <= vpoLatch;
                        write_addr <= grab_addr;
                end if;
                if clr_grab_cntr = '1' then
                        grab_cntr_hori <= (others => '0');
                        grab_cntr_vert <= (others => '0');
                else
                        if (nueva_muestra = '1' and muestrablue = '1')  then
                                grab_cntr_hori <= grab_cntr_hori + 1;
                        end if;
                        if inc_grab_vert = '1' then
                                        grab_cntr_vert <= grab_cntr_vert + 1;
                                grab_cntr_hori <= (others => '0');
                        end if;
                end if;
        end if;
end process control_escrituras;

--Formacion de direccion de escritura mediantre concatenacion de contador
--horizontal y vertical de muestras de entrada
grab_addr <= '0' & grab_cntr_vert(8 downto 0) & grab_cntr_hori(8 downto 0);

--Registro que almacena la primera lectura de SRAM
registro_lectura_1:process(clk27, rstn)
begin
        if rstn = '0' then
                pixeltmp1 <= (others => '0');
        elsif clk27'event and clk27 = '1' then
                if doingRead1 = '1' then
                        pixeltmp1 <= SRAMData;
```

```
                end if;
        end if;
end process registro_lectura_1;

--Registro quie almacena la segunda lectura de SRAM
registro_lectura_2:process(clk27, rstn)
begin
        if rstn = '0' then
                pixeltmp2 <= (others => '0');
        elsif clk27'event and clk27 = '1' then
                if doingRead2 = '1' then
                        pixeltmp2 <= SRAMData;
                end if;
        end if;
end process registro_lectura_2;

--Multiplex que selecciona los pixeles adecuados del buffer dependiendo
--del barrido de pantalla por medio del modulo VGACORE
mux_pixel:
with hloc(1 downto 0) select
                pixelint<=      pixeltmp1(15 downto 8) when "00",
                   pixeltmp1(7 downto 0) when "01",
                                        pixeltmp2(15 downto 8) when "10",
                                        pixeltmp2(7 downto 0) when "11",
                                        "00000000" when others;

--Puerto de pixel hacia RAMDAC registrado
registro_pixel:process(clk27, rstn)
begin
        if rstn='0' then
                pixel <= (others => '0');
        elsif clk27'event and clk27='1' then
                pixel <= pixelint;
        end if;
end process registro_pixel;

--Actualizacion de senales de FSM de gestion memoria
control_Ram:process(clk27, rstn)
begin
        if rstn = '0' then
                presStateRAM <= stWriteS1;
                doingWrite1 <= '0';
                doingWrite2 <= '0';
                doingRead1 <= '0';
                doingRead2 <= '0';
        elsif clk27'event and clk27 = '1' then
                presStateRAM <= nextStateRAM;
                doingWrite1 <= nextdoingWrite1;
                doingWrite2 <= nextdoingWrite2;
                doingRead1 <= nextdoingRead1;
                doingRead2 <= nextdoingRead2;
        end if;
end process control_Ram;

--Proceso que secuencia y alterna los accesos a SRAM y genera las
--direcciones de lectura
control_Ram_FSM:process(presStateRAM, vloc, hlocint)
begin

        nextStateRAM <= presStateRAM;
        nextdoingWrite1 <= '0';
        nextdoingWrite2 <= '0';
        nextdoingRead1 <= '0';
        nextdoingRead2 <= '0';
        read_addr <= '0' & vloc(9 downto 1) & hlocint & '0';

        case presStateRAM is

                when stWriteS1 =>
                                nextStateRAM <= stWriteS2;
                                nextdoingWrite1 <= '1';

                when stWriteS2 =>
                                nextStateRAM <= stReadS1;
                                nextdoingWrite2 <= '1';
```

```
                when stReadS1 =>
                        nextStateRAM <= stReadS2;
                        nextdoingRead1 <= '1';
                        read_addr <= '0' & vloc(9 downto 1) & hlocint & '0';

                when stReadS2 =>
                        nextStateRAM <= stWriteS1;
                        nextdoingRead2 <= '1';
                        read_addr <= '0' & vloc(9 downto 1) & hlocint & '1';

        end case;
end process control_Ram_FSM;

--Generamos un contador horizontal interno ya que se trabaja en bloques de
--16 bits que corresponden a 2 pixeles en pantalla
registro_hdir:process(clk27, rstn)
begin

        if rstn='0' then
                hlocint <= (others => '0');
        elsif clk27'event and clk27='1' then
                hlocint <= hloc(9 downto 2);
        end if;
end process registro_hdir;

--Parte del bus de datos bidireccional que realiza las escrituras
--Cuando no escribe deja el bus en alta impedancia para permitir su lectura
registro_escritura:process(doingWrite2, muestra1)
begin

        if      (doingWrite2 = '1') then
                SRAMData <= muestra1;
        else
                SRAMData <= (others => 'Z');
        end if;
end process registro_escritura;

--Senal que habilita la escritura en memoria
control_escritura:process(doingWrite2)
begin

        if      (doingWrite2 = '1') then
                WEn <= '0';
        else
                WEn <= '1';
        end if;
end process control_escritura;

--Senal que permite la lectura de memoria
control_lectura:process(doingRead1, doingRead2)
begin

        if      (doingRead1 = '1' or doingRead2='1') then
                OEn <= '0';
        else
                OEn <= '1';
        end if;
end process control_lectura;

end videoproc_arch;
```

```
----------------------------------------------------------------------------------------------------------------
--Modulo SAA7113.VHD
--Modulo de mas alto nivel en el diseño, integra como componentes todos los
--elementoa anteriormente descritos, gestiona la rutina de inicializacion de
--dispositivo como RAMDAC y decoder video.
----------------------------------------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity saa7113 is
port(
        clk: in std_logic;                                              --Reloj de sistema 50MHz
--      lp_d: in std_logic_vector(2 downto 0);  --Interfaz con puerto paralelo
--      lp_s: out std_logic_vector(3 downto 3);
        sw: in std_logic_vector(3 downto 1);    --switches de reset y pause
        scl: inout std_logic;                                   --reloj de i2c
        sda: inout std_logic;                                   --datos de i2c
        llck: in std_logic;                                             --reloj de decoder 27MHz
        vpo: in std_logic_vector(7 downto 0);   --datos de video
--      rts: in std_logic_vector(1 downto 0);
        hsyncb: buffer std_logic;                               --sincronismo horizontal
        vsyncb: out std_logic;                                  --sincronismo vertical
        blankb: out std_logic;                                  --señal de borrado
        trste: out std_logic;                                   --enable del controlador ethernet
        bar: out std_logic_vector(7 to 9);              --led de estado
        p: out std_logic_vector(7 downto 0);    --pixeles hacia RAMDAC
        pixelclk: out std_logic;                                --reloj de pixel 27MHz
        wrb, rdb: out std_logic;                                --senales de control RAMDAC
        d: inout std_logic_vector(7 downto 0);  --Puerto configuracion RAMDAC
        rs: inout std_logic_vector(2 downto 0); --Seleccion registro RAMDAC
        rceb, lceb, loeb, lweb: out std_logic;  --Senales control memoria
        la: out std_logic_vector(18 downto 0);  --Bus de direcciones de memoria
        ld: inout std_logic_vector(15 downto 0) --Bus de datos de memoria
);
end saa7113;

architecture saa7113_arch of saa7113 is

--Componente de libreria Xilinx que implementa un buffer global,
--tipicamente dedicado a senales de reloj.
component bufg
port(
        I: in std_logic;
        O: out std_logic
);
end component;

--Cpnfiguracion de RAMDAC
component prgramdacver2
port(
    clk: in std_logic;
    rstn: in std_logic;
    start: in std_logic;
    done: out std_logic;
    WRn: out std_logic;
    RDn: out std_logic;
    RS: inout std_logic_vector (2 downto 0);
    data: inout std_logic_vector (7 downto 0)
);
end component;

--Procesador de video
component videoprocess
port(
        clk27: in std_logic;
        vpo: in std_logic_vector(7 downto 0);
        rstn: in std_logic;
        capture: in std_logic;
        pixel: out std_logic_vector(7 downto 0);
        pixelclk: out std_logic;
        hsync: buffer std_logic;
        vsync: out std_logic;
        enable: out std_logic;
        lceb, loeb, lweb: out std_logic;
```

```
		addmem: out std_logic_vector(18 downto 0);
		datamem: inout std_logic_vector(15 downto 0);
		capturando: out std_logic
);
end component;

--Eliminador de rebotes para el congelado de imagen
component debouncer
port(
   rst: in std_logic;
   clk: in std_logic;
   x: in std_logic;
   xDeb: out std_logic;
   xDebFallingEdge: out std_logic;
   xDebRisingEdge: out std_logic
  );
end component;

--Configuraciionjj de SAA7113 mediante I2C
component i2c
port(
		clk: in std_logic;
		rst: in std_logic;
		start: in std_logic;
		done: out std_logic;
		scl: out std_logic;
		sda: out std_logic
);
end component;


type VGA_STATE_TYPE is (stReset, stWaste1, stWaste2, stWaste3, stWait, stForever);

signal presStateVGA: VGA_STATE_TYPE;
signal startProg : std_logic;
signal startVGA : std_logic;
signal done,donei2c,doneramdac : std_logic;
signal rstn: std_logic;
signal clk27: std_logic;
signal gnd_net : std_logic;
signal capture : std_logic;
signal global_clk:std_logic;

begin

ramdac_init: prgramdacver2
port map(
		clk => global_clk,
		rstn => rstn,
		start => startProg,
		done => doneRamDac,
		WRn => wrb,
		RDn => rdb,
		RS => rs,
		data => d
);

video_processor: videoprocess
port map(
		clk27 => clk27,
		vpo => vpo,
		rstn => rstn,
		capture => capture,
		pixel => p,
		pixelclk => pixelclk,
		hsync => hsyncb,
		vsync => vsyncb,
		enable => blankb,
		lceb => lceb,
		loeb => loeb,
		lweb => lweb,
		addmem => la,
		datamem => ld,
		capturando => bar(8)
);
```

```
debouncer_inst:debouncer
port map(
   rst => rstn,
   clk => clk27,
   x => sw(2),
   xDeb => capture,
   xDebFallingEdge => open,
   xDebRisingEdge => open
);

ub: bufg port map(I=>llck,O=>clk27);                --buffer global para reloj de video
ub2: bufg port map(I=>clk,O=>global_clk);           --buffer global para reloj de sistema

i2c_inst:i2c
port map(
            clk => global_clk,
            rst => rstn,
            start => startProg,
            done => donei2c,
            scl => scl,
            sda => sda
);

--Inicializacion terminada cuando terminan RAMDAC y Decoder de video
done <= donei2c and doneramdac;

--Reset global
rstn <= sw(1);
gnd_net <= '0';
--Triestado del controlador ethernet por comparticion de buses
trste <= '1';
--Triestado del banco derecho de SRAM por conflictos con RAMDAC
rceb <= '1';
bar(7) <= not done;

inicializacion_sistema:process(clk27, rstn)
            begin
                        if rstn = '0' then
                                    presstateVGA <= stReset;
                        elsif clk27'event and clk27 = '1' then
                                    case presstateVGA is

                                                when stReset =>
                                                            presstateVGA <= stWaste1;

                                                when stWaste1 =>
                                                            presstateVGA <= stWaste2;

                                                when stWaste2 =>
                                                            presstateVGA <= stWaste3;

                                                when stWaste3 =>
                                                            presstateVGA <= stWait;

                                                when stWait =>
                                                            if done = '0' then
                                                                        presstateVGA <= stWait;
                                                            else
                                                                        presstateVGA <= stForever;
                                                            end if;

                                                --Una vez inicializado queda en este estado hasta un reset
                                                when stForever =>
                                                            presstateVGA <= stForever;
                                    end case;
                        end if;
end process inicializacion_sistema;

inic_sis_FSM:process(presStateVGA)
begin
            startProg <= '0';
            startVGA <= '0';

            case presstateVGA is

                        when stReset =>
```

```
                        startProg <= '0';
                        startVGA <= '0';

                when stWaste1 =>
                        startProg <= '1';
                        startVGA <= '0';

                when stWaste2 =>
                        startProg <= '1';
                        startVGA <= '0';

                when stWaste3 =>
                        startProg <= '1';
                        startVGA <= '0';

                when stWait =>
                        startProg <= '1';
                        startVGA <= '0';

                when stForever =>
                        startProg <= '1';
                        startVGA <= '1';
        end case;
end process inic_sis_FSM;

end saa7113_arch;
```

```
#------------------------------------------------------------------------------------------------------------------
#VIDEOIN.UCF
#Constraint file
#Fichero donde se especifica lasignacion de pines de la FPGA con las señales del diseño
#------------------------------------------------------------------------------------------------------------------

#
# pin assignments for the Virtex chip on the XSV Board
#

# Flash RAM
# net frdy                loc=p171;
# net fceb                loc=P170;
# net foeb                loc=p173;
# net fweb                loc=p131;
# net fd<0>               loc=p177;
# net fd<1>               loc=p167;
# net fd<2>               loc=p163;
# net fd<3>               loc=P156;
# net fd<4>               loc=P145;
# net fd<5>               loc=P138;
# net fd<6>               loc=P134;
# net fd<7>               loc=P124;
# net fa<0>               loc=p132;
# net fa<1>               loc=p133;
# net fa<2>               loc=p139;
# net fa<3>               loc=p141;
# net fa<4>               loc=p144;
# net fa<5>               loc=p147;
# net fa<6>               loc=p152;
# net fa<7>               loc=p154;
# net fa<8>               loc=p157;
# net fa<9>               loc=p160;
# net fa<10>              loc=p162;
# net fa<11>              loc=p169;
# net fa<12>              loc=p168;
# net fa<13>              loc=p161;
# net fa<14>              loc=p159;
# net fa<15>              loc=p155;
# net fa<16>              loc=p153;
# net fa<17>              loc=p149;
# net fa<18>              loc=p146;
# net fa<19>              loc=p142;
# net fa<20>              loc=p140;

# parallel port interface through CPLD
#net lp_d<0>             loc=p177;
#net lp_d<1>             loc=p167;
#net lp_d<2>             loc=p163;
# net lp_d<3>             loc=p156;
# net lp_d<4>             loc=p145;
# net lp_d<5>             loc=p138;
# net lp_d<6>             loc=p134;
# net lp_d<7>             loc=p124;
#net lp_s<3>             loc=p132;
# net lp_s<4>             loc=p133;
# net lp_s<5>             loc=p139;
# net lp_s<6>             loc=p141;

# DIP and pushbutton switches
# net dipsw<1>            loc=p161;
# net dipsw<2>            loc=p159;
# net dipsw<3>            loc=p155;
# net dipsw<4>            loc=p153;
# net dipsw<5>            loc=p149;
# net dipsw<6>            loc=p146;
# net dipsw<7>            loc=p142;
# net dipsw<8>            loc=p140;
net sw<1>                 loc=p174;
net sw<2>                 loc=p175;
#net sw<3>                loc=p176;
# net sw<4>               loc=p185;

# left, right, and bargraph LEDs
# net ls<0>               loc=p177;
# net ls<1>               loc=p167;
```

```
# net ls<2>          loc=p163;
# net ls<3>          loc=p156;
# net ls<4>          loc=p145;
# net ls<5>          loc=p138;
# net ls<6>          loc=p134;
# net rs<0>          loc=p124;
# net rs<1>          loc=p132;
# net rs<2>          loc=p133;
# net rs<3>          loc=p139;
# net rs<4>          loc=p141;
# net rs<5>          loc=p144;
# net rs<6>          loc=p147;
# net bar<0>         loc=p152;
# net bar<1>         loc=p154;
# net bar<2>         loc=p157;
# net bar<3>         loc=p160;
# net bar<4>         loc=p162;
# net bar<5>         loc=p169;
#net bar<6>          loc=p168;
net bar<7>           loc=p173;
net bar<8>           loc=p131;
#net bar<9>          loc=p171;

# Ethernet transceiver
# net fds_mdint      loc=p18;
# net mdc            loc=p19;
# net mdio           loc=p20;
# net crs            loc=p21;
# net col            loc=p23;
net trste            loc=p24;
# net txen           loc=p25;
# net rxdv           loc=p26;
# net txerr          loc=p27;
# net rxerr          loc=p28;
# net txd<4>         loc=p31;
# net txd<3>         loc=p39;
# net txd<2>         loc=p40;
# net txd<1>         loc=p41;
# net txd<0>         loc=p42;
# net rxd<4>         loc=p33;
# net rxd<3>         loc=p34;
# net rxd<2>         loc=p35;
# net rxd<1>         loc=p36;
# net rxd<0>         loc=p38;
# net txclk          loc=p210;
# net rxclk          loc=p213;

# video decoder
net llck             loc=p92;
#net rts<1>          loc=p110;
#net rts<0>          loc=p111;
# net rtc0           loc=p113;
net scl              loc=p114;
net sda              loc=p115;
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;

# video RAMDAC
net rs<2>            loc=p26;
net rs<1>            loc=p28;
net rs<0>            loc=p31;
net d<0>             loc=p42;
net d<1>             loc=p41;
net d<2>             loc=p40;
net d<3>             loc=p39;
net d<4>             loc=p38;
net d<5>             loc=p36;
net d<6>             loc=p35;
net d<7>             loc=p34;
net wrb              loc=p46;
```

```
net rdb                 loc=p47;
net hsyncb              loc=p48;
net vsyncb              loc=p49;
net blankb              loc=p50;
net pixelclk            loc=p52;
net p<0>                loc=p70;
net p<1>                loc=p71;
net p<2>                loc=p72;
net p<3>                loc=p73;
net p<4>                loc=p74;
net p<5>                loc=p78;
net p<6>                loc=p79;
net p<7>                loc=p80;

# stereo codec
# net mclk              loc=p3;
# net lrck              loc=p4;
# net sclk              loc=p5;
# net sdin              loc=p6;
# net sdout             loc=p7;

# left SRAM bank
net lceb                oc=p186;
net loeb                loc=p228;
net lweb                loc=p201;
net ld<0>               loc=p202;
net ld<1>               loc=p203;
net ld<2>               loc=p205;
net ld<3>               loc=p206;
net ld<4>               loc=p207;
net ld<5>               loc=p208;
net ld<6>               loc=p209;
net ld<7>               loc=p215;
net ld<8>               loc=p216;
net ld<9>               loc=p217;
net ld<10>              loc=p218;
net ld<11>              loc=p220;
net ld<12>              loc=p221;
net ld<13>              loc=p222;
net ld<14>              loc=p223;
net ld<15>              loc=p224;
net la<0>               loc=p200;
net la<1>               loc=p199;
net la<2>               loc=p195;
net la<3>               loc=p194;
net la<4>               loc=p193;
net la<5>               loc=p192;
net la<6>               loc=p191;
net la<7>               loc=p189;
net la<8>               loc=p188;
net la<9>               loc=p187;
net la<10>              loc=p238;
net la<11>              loc=p237;
net la<12>              loc=p236;
net la<13>              loc=p235;
net la<14>              loc=p234;
net la<15>              loc=p232;
net la<16>              loc=p231;
net la<17>              loc=p230;
net la<18>              loc=p229;

# right SRAM bank
net rceb                loc=p109;
#net roeb               loc=p95;
#net rweb               loc=p68;
#net rd<0>              loc=p70;
#net rd<1>              loc=p71;
#net rd<2>              loc=p72;
#net rd<3>              loc=p73;
#net rd<4>              loc=p74;
#net rd<5>              loc=p78;
#net rd<6>              loc=p79;
#net rd<7>              loc=p80;
#net rd<8>              loc=p81;
#net rd<9>              loc=p82;
#net rd<10>             loc=p84;
```

```
#net rd<11>             loc=p85;
#net rd<12>             loc=p86;
#net rd<13>             loc=p87;
#net rd<14>             loc=p93;
#net rd<15>             loc=p94;
#net ra<0>              loc=p67;
#net ra<1>              loc=p66;
#net ra<2>              loc=p65;
#net ra<3>              loc=p64;
#net ra<4>              loc=p63;
#net ra<5>              loc=p57;
#net ra<6>              loc=p56;
#net ra<7>              loc=p55;
#net ra<8>              loc=p54;
#net ra<9>              loc=p53;
#net ra<10>             loc=p108;
#net ra<11>             loc=p107;
#net ra<12>             loc=p103;
#net ra<13>             loc=p102;
#net ra<14>             loc=p101;
#net ra<15>             loc=p100;
#net ra<16>             loc=p99;
#net ra<17>             loc=p97;
#net ra<18>             loc=p96;

# PS/2 connector
# net ps2clk            loc=p13;
# net ps2data           loc=p17;

# USB transceiver
# net usbvm             loc=p9;
# net usbvp             loc=p10;
# net usbrcv            loc=p11;
# net usboeb            loc=p12;
# net usbvpo            loc=p13;
# net usbvmo            loc=p17;

# programmable oscillator
net clk                 loc=p89;
```