UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE INGENIERÍA

Diseño de una Cámara Inteligente

TESIS

que presenta para obtener el grado de MAESTRO EN INGENIERÍA

Iván Olaf Hernández Fuentes

DIRECTOR DE TESIS:

Dr. Miguel Enrique Bravo Zanoguera

Co-Director

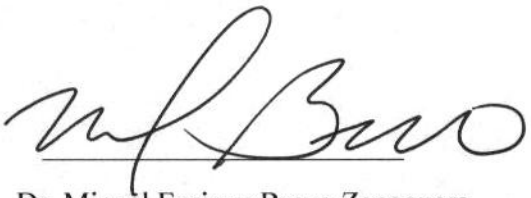
M.C. Guillermo Galaviz Yañez

MEXICALI, B. C. **ENERO 2008**

RESUMEN de la Tesis de Iván Olaf Hernández Fuentes, presentada como requisito parcial para la obtención del grado de MAESTRO EN INGENIERÍA. Mexicali, Baja California, México, Enero de 2008.

DISEÑO DE UNA CÁMARA INTELIGENTE

Resumen aprobado por:



Dr. Miguel Enrique Bravo Zanoguera
Director de tesis

Se presenta el diseño de un dispositivo inteligente para la captura de imágenes, además de capturar la imagen de una escena puede realizar procesamiento a los datos capturados sin la intervención de un CPU. La arquitectura se basa en un sensor CMOS de imagen de bajo costo y de alto desempeño, Micron MT9T001 como elemento de captura, y como procesador de los datos de los píxeles se emplea un dispositivo lógico reconfigurable FPGA Cyclone II de Altera. En este trabajo se implementan operaciones de procesamiento de imagen de bajo nivel sobre la salida digital cruda del sensor de imagen CMOS (con patrón de color de Bayer). Utilizando una arquitectura de procesamiento en "pipeline" (segmentación) en el FPGA resulta una alta tasa de datos, ya que después de una latencia inicial requerida para almacenar dos líneas de píxeles del cuadro de imagen, se alcanza a calcular cada píxel a la misma velocidad de entrada de datos, lo que nos brinda resultados en tiempo real, pues no se necesita esperar que se almacene un cuadro completo para obtener resultados. Como producto de cámara inteligente, se desarrolló un sistema embebido que incluye: comunicación I2C con el sensor, captura digital, procesamiento de interpolación para el desmosaico del patrón de bayer, corrección de color y despliegue de imagen (formato VGA). Con un solo dispositivo FPGA se implementaron el conjunto de subsistemas digitales para realizar las distintas operaciones. El diseño mantiene las funciones sofisticadas de cámara, nativas (on-chip) del sensor CMOS, tales como modificar los tamaños de cuadros, tasas de cuadro, y otras, que permiten hacer mas versátil las posibles aplicaciones de este producto. Los recursos utilizados del FPGA para implementar esta tesis representan un porcentaje bajo comparado con la capacidad total del FPGA, por lo que quedan recursos disponibles para realizar otras funciones de procesamiento o para integrar otro tipo de conectividad. Además, en esta tesis se desarrolló un ambiente de trabajo propio, donde se utilizaron los elementos hardware y software que se consideraban lo mínimo necesario para formar un ambiente controlado de diseño y verificación de datos. Este ambiente de desarrollo para diseño de sistemas digitales embebidos puede ser utilizado en aplicaciones que requieran procesamiento de video a bajo costo y sistemas de visión que requieren de portabilidad, y puede cumplir con funciones más complejas de procesamiento de imagen en aplicaciones especificas de visión por maquina o inspección industrial.

Agradecimientos

A mis padres, familiares y amigos por su comprensión y apoyo.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT), por brindarme el apoyo económico necesario para lograr la realización de mis estudios.

A Miguel Bravo Zanoguera y Guillermo Galaviz Yañez por su guía durante el desarrollo de esta tesis.

Índice de figuras

Capítulo I

Introducción

1.1 Antecedentes de Cámara Inteligente

Cuando escuchamos la palabra cámara inteligente es posible que el término nos resulte ambiguo, es decir, ¿a qué se refiere con inteligente? El término "inteligente" es comúnmente usado en el ámbito comercial donde se habla de sensores inteligentes y tarjetas inteligentes; sin embargo estos dispositivos no cumplen necesariamente con la definición de inteligencia artificial sino que son dispositivos que nos aportan algún servicio adicional. Por ejemplo, se habla de sensores de imagen inteligentes a aquellos dispositivos que además de capturar la imagen de una escena pueden realizar algún tipo de procesamiento a los datos capturados.

Dentro de la bibliografía de trabajos de investigación sobre cámara inteligente, algunas publicaciones recientes consideran la "inteligencia" en medida del apoyo que brindan dichos dispositivos en la toma de decisiones para aplicaciones en sistemas automáticos. Como en el caso de [1] donde se desarrolla una cámara inteligente para reconocimiento de gestos simples de mano y cara, la arquitectura de dicha cámara esta basada en el uso de un procesador reconfigurable implementado completamente en un solo dispositivo FPGA (Arreglo de Compuertas Programado en Campo) y el uso de un sensor de imagen CMOS (Semiconductor de Metal Oxido Complementario) o en [3] donde se desarrolla también una cámara inteligente para reconocimiento de gestos, donde usaron como arquitectura base cámaras comerciales con salida analógica, tarjeta de video VLSI y computadora personal

Existen otros trabajos de investigación que persiguen desarrollar un ambiente, donde el diseñador pueda moldear su propia cámara inteligente según su aplicación. Ejemplos de lo anterior son [2] donde se propone una metodología de diseño para crear ASICs (Circuitos Integrados de Aplicación Especifica) para cámaras inteligentes y en [4] (que es seguimiento de 2) en donde se estudia un conjunto de procesadores de propósito general para cámaras inteligente trabajando tanto con funciones de imagen de

bajo y alto nivel, la arquitectura base de esta plataforma de desarrollo esta compuesta por un sensor CMOS de imagen, Lógica reconfigurable en FPGA y procesadores LPA (Arreglo de Procesadores Lineales) y ILP (Procesadores Paralelos a Nivel de Instrucción), en dicha arquitectura se puede ingresar parámetros como resolución, número de elementos procesadores y funcionalidad de los mismos.

1.2 Introducción del trabajo realizado en esta tesis

Esta tesis presenta el diseño de una arquitectura de cámara inteligente basada en un dispositivo digital reconfigurable de bajo costo y de alto desempeño de procesamiento. Esta arquitectura consta de un sensor CMOS de imagen, Micron MT9T001, como elemento de captura. Como procesador de los datos de los píxeles se emplea un dispositivo reconfigurable FPGA Cyclone II de Altera, dadas sus características de velocidad y versatilidad de implementar cualquier tipo de sistema digital.

Este trabajo se limita a implementar operaciones de procesamiento de imagen de bajo nivel[1], que pueden ser mapeadas a un dispositivo reconfigurable FPGA utilizando operaciones en "pipeline" (segmentación), lo que proporciona una alta tasa de datos ya que después de una latencia inicial se alcanza a calcular cada píxel a la frecuencia de entrada de datos lo que nos brinda resultados en tiempo real. Además, el uso de este dispositivo permite que el sistema sea portátil para cumplir con las restricciones de posibles aplicaciones tales como visión por maquina o inspección industrial; este trabajo aportará al desarrollo de tecnología propia tomando en cuenta que futuros proyectos de nuestro laboratorio tengan como punto de partida este trabajo para implementar funciones más complejas de imagen para aplicaciones especificas.

Para configurar los diseños en un dispositivo FPGA existen herramientas de Diseño Asistido por Computadora (CAD) en las cuales se ingresa un modelo de forma esquemática o con un lenguaje de descripción de hardware (HDL). Se decidió en esta tesis utilizar el lenguaje VHDL (Lenguaje de Descripción de Hardware para Circuitos

[1] Se les llama de bajo nivel ya que son operaciones matemáticas con estructuras bien definidas y que se aplican de forma repetitiva todos los píxeles de una imagen dando como resultado una imagen de salida.

Integrados de muy alta Velocidad) el cual es un estándar del IEEE (Instituto de Ingenieros Eléctricos y Electrónicos), ya que de esta manera se podrán implementar los códigos realizados en cualquier dispositivo FPGA independientemente del fabricante o de la herramienta CAD utilizada.

También cabe mencionar que el diseño de esta plataforma se hizo desde un principio ("from the scratch") es decir; que la etapa de captura de imagen no esta basada en una cámara comercial que otorgue una salida analógica o digital procesada como en otros trabajos, sino que se utiliza la salida digital cruda (con el patrón de color de bayer) del sensor de imagen CMOS de tal forma que se aplicó nuestro propio procesamiento para obtener el color y se demuestra que algún otro tipo de algoritmos que sea necesario se puede implementar.

El desarrollo de una aplicación depende principalmente de dos factores, el ambiente de la plataforma de desarrollo y del lenguaje de programación. En esta tesis se desarrolló un ambiente de trabajo propio, donde se utilizaron los elementos hardware y software que se consideraban lo mínimo necesario para formar un ambiente controlado de adquisición y verificación de datos, se evaluaron varias opciones disponibles, para buscar lo que nos pudiera dar el mejor control del desarrollo a bajo costo. Se trabajó con dispositivos de NI y se utilizaron los lenguajes de Matlab y Visual C++ para transferir datos desde la PC, en la etapa de simulación de datos. Por otra parte para el diseño del sistema de imagen en FPGA, se utilizaron tarjetas de desarrollo UP2, DE2 y software QUARTUSII de Altera. Los resultados de las diferentes configuraciones de plataforma de desarrollo son presentados.

Como producto de cámara inteligente, se desarrolló un sistema completo de captura, procesamiento y despliegue de imagen totalmente embebido[2] donde con un solo dispositivo FPGA se implementaron el conjunto de subsistemas digitales para realizar las distintas operaciones entre las que se pueden mencionar:

[2] El termino embebido se refiere a que el sistema tiene la capacidad de funcionar sin la ayuda de la computadora

- Modulo para escribir comandos al sensor CMOS de imagen controlando sus funciones, en el cual se implemento el protocolo de comunicación I2C.
- Módulo que permitió desplegar en un monitor VGA una imagen fija proveniente de la PC.
- Módulo que permitió el despliegue de un cuadro capturado por el sensor CMOS (tipo cámara "still").
- Módulo que permitió el despliegue en tiempo real de los datos provenientes del sensor en un solo canal de color.
- Módulo para aplicar la interpolación bilineal necesaria para obtener los 3 componentes de color del sensor CMOS (el cual tiene un patrón de color Bayer).
- Módulo que permitió el despliegue de los datos interpolados en los 3 canales de color.
- Módulo que aplica una matriz de corrección de color para buscar que los colores se desplieguen en el monitor de una manera que sean percibidos agradablemente por el ojo humano.

Con lo antes mencionado se alcanzaron los objetivos siguientes: Producir una cámara inteligente que conste de un sensor de imagen de tipo CMOS y un procesador reconfigurable hecho en FPGA. Crear una biblioteca de funciones en VHDL para captura, procesamiento y despliegue de imagen, capacidad de comunicación con dispositivos que acepten protocolo I2C. Logrando un sistema completo que aplica la configuración del sensor, la interpolación bilineal, la corrección de color y el despliegue en tiempo real en los 3 canales de color.

En la figura 1.1 se muestra un diagrama breve de las partes que conforma el sistema diseñado en esta tesis, se puede mencionar que en el bloque de procesamiento de imagen se implementa la estructura en pipeline[3] para aplicar la interpolación bilineal y que es posible implementar cualquier operación de imagen que siga esta misma, como el caso de los filtros lineales que utilizan operaciones de vecindad.

[3] El termino pipeline se conoce en español como "segmentación" en lo textos de arquitectura de computadora y consiste en dividir una tarea en subtareas que pueden realizarse de forma independiente es decir, no tiene que esperar que termine una para que empiece otra .Es un intento por paralelizar operaciones secuenciales.

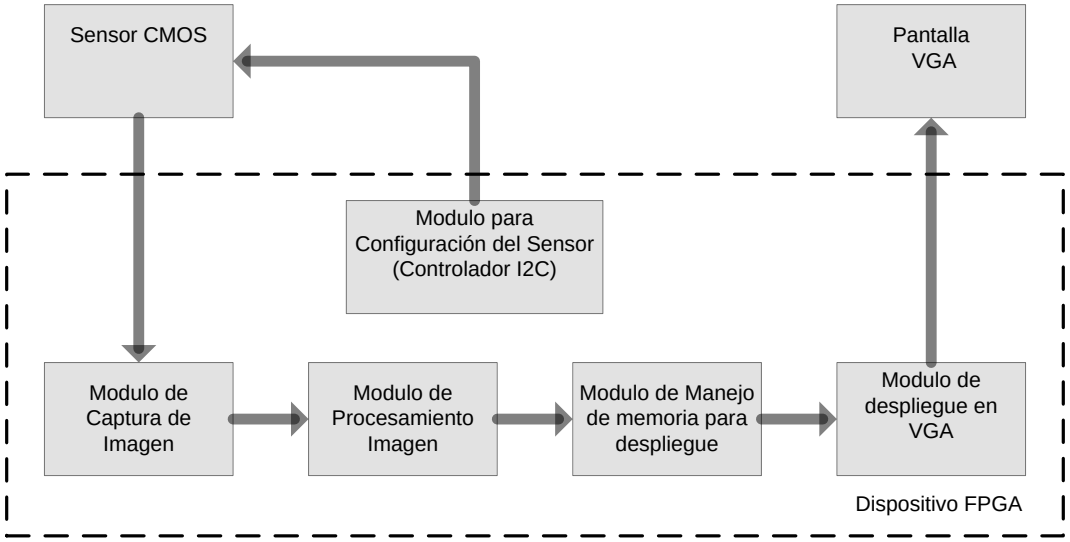


Figura 1.1 Esquema simplificado de la arquitectura desarrollada en esta tesis

El capítulo 1 fue una breve introducción a los antecedentes de cámara inteligente y al trabajo realizado en esta tesis. En el capitulo 2 se expone la teoría básica necesaria para entender el funcionamiento de los sensores CMOS de imagen y los dispositivos reconfigurables FPGA. En el capítulo 3 se habla de lleno de la metodología utilizada para el diseño del sistema digital y de los componentes que conforman la plataforma de trabajo desarrollada. En el capitulo 4 se discutirán los resultados obtenidos tanto de la plataforma de desarrollo como del diseño digital implementado en FPGA. En el capítulo 5 se muestra la conclusión de la tesis y se habla del trabajo futuro. Además se anexan tres apéndices que incluyen los códigos realizados en VHDL, Matlab y Visual C++.

Capítulo II

Marco Teórico

2.1 Sensor de imagen CMOS

2.1.1 Teoría de los sensores de imagen CMOS

Tradicionalmente el uso de la tecnología de sensores de imagen de dispositivo de carga acoplada (CCD por sus siglas en inglés) ha sido la preferente, en años recientes los sensores de imagen CMOS han demostrado igualar la calidad de los CCD, añadiendo ciertas ventajas en los diseños.

Los sensores de imagen cuentan con un área de trabajo en la que se encuentra un arreglo de píxeles (arreglo de dos dimensiones), cada uno contiene un fotodetector que convierte la luz incidente en foto corriente y algunos de los circuitos requeridos para convertir la foto corriente en carga eléctrica o voltaje para leerla fuera del arreglo. (El porcentaje de área ocupado por el fotodetector en un píxel se le conoce como "fill factor").

Los sensores CCD transfieren las cargas captadas por sus píxeles de un renglón a otro hasta que se llega al registro de lectura que alimenta un amplificador que convierte la carga en voltaje alimentando, después un convertidor analógico digital enviando los datos de salida serialmente. En los sensores CMOS las señales de carga son leídas un renglón a la vez de manera similar a una memoria RAM, usando circuitos de selección de renglón y columna. La ventaja de la arquitectura de lectura del CCD es que requiere un mínimo de circuitos, haciendo posible diseñar sensores de imagen con tamaños de píxel muy pequeños, sin embargo la lectura de la transferencia de carga es serial limitando la velocidad de salida además tiene la necesidad de usar relojes de muy altas tazas y altos voltajes para lograr la transferencia adecuada por lo que la demanda de potencia se incremente. En los sensores CMOS la lectura de acceso aleatorio provee lecturas de alta velocidad y operaciones de ventana de interés a bajo consumo de potencia, esto hace que los CMOS sean adecuados para implementar imagen de muy alta resolución especialmente para aplicaciones de video. En la figura 2.1 se puede observar la arquitectura de lectura de los sensores CMOS.

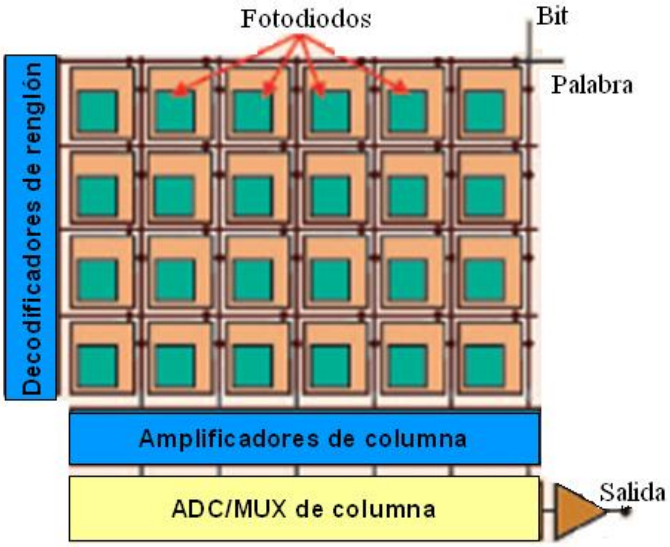


Figura 2.1 Arquitectura de lectura de los sensores CMOS

La tecnología utilizada para fabricar los sensores CCD es especializada para dispositivos de imagen permitiendo escalar el tamaño del píxel más pequeños sin degradar la calidad sin embargo, no permite integrar otras funciones (circuitos) dentro del mismo chip. Los sensores CMOS son fabricados en su mayoría con tecnología estándar por lo que se pueden integrar otros circuitos de procesamiento analógico, digital y de control, esto también reduce la potencia y el tamaño del sistema.

Explotando la habilidad de integrar sensado con procesamiento analógico y digital al nivel de píxel, nuevos tipos de dispositivos de imagen CMOS están siendo creados para interfaces máquina, vigilancia y monitoreo, visión por máquina, pruebas biológicas, entre otras aplicaciones [5].

La arquitectura de píxel más usada actualmente en los sensores CMOS, es la de sensores de píxel activos (APS) que usan 3 o 4 transistores dentro del píxel, con el propósito de incrementar considerablemente la velocidad de lectura y mejorar la razón señal a ruido.

En ciertas aplicaciones como en sistemas de imagen móviles existe la necesidad de incrementar la resolución espacial sin incrementar el área del sensor, es decir se necesitan píxeles más pequeños, en este aspecto los sensores CCD tienen ventaja. En los sensores CMOS se está aprovechado la elevación en la escala de integración de su tecnología además de usar nuevas arquitecturas de píxel que reducen el efecto del número de píxeles compartiendo algunos transistores entre grupos de píxeles vecinos.

2.1.2 El sensor de imagen CMOS MT9T001

El M9T001 es un sensor de imagen CMOS de píxel activo con un arreglo de píxeles de 2,048 horizontal por 1,536 vertical (formato QXGA) y un formato óptico de ½ pulgada. Cuenta con un arreglo de filtro de color con patrón de Bayer. Cuenta con un convertidor analógico digital dentro del chip que provee 10 bits por píxel. Tiene una tasa de píxel máxima de 48 megapíxeles por segundo es decir, que soporta un reloj maestro máximo de 48 Mhz. Se alimenta con fuente de voltaje de 3.0v-3.6v (3.3 nominal), tiene un consumo de potencia de 240mW (nominal). Posee tecnología CMOS que alcanza la calidad de la CCD mientras mantiene las ventajas de tamaño inherente, costo e integración de CMOS. El sensor puede ser operado en su modo predeterminado o programado por el usuario para el tamaño de cuadro, exposición, ajustes de ganancia y otros parámetros. Se programa a través de una interfase sencilla de dos alambres (I2C). El modo predeterminado entrega una salida de imagen en formato QXGA a 12 cuadros por segundo (fps). Las señales FRAME_VALID y LINE_VALID son pins dedicados de salida junto con una señal de reloj de píxel PXCLK que está en sincronía con los datos válidos. Tiene la capacidad de capturar cuadros continuos de video y cuadros sencillos, lo que lo hace una opción excelente para varias aplicaciones como cámaras "still", cámaras de video y cámaras de PC. En la figura 2.2 se muestra un diagrama a bloques del sensor

El MT9T001 usa un arreglo de filtro de color (CFA) con el patrón de color bayer donde, cada píxel tiene un filtro de uno de los colores primarios como se muestra en la figura 2.3, los renglones pares contienen píxeles de color verde y rojo y los impares contiene píxeles de color azul y verdes. Las columnas pares contiene píxeles de color verde y azul; las columnas impares contienen píxeles de color rojo y verde.

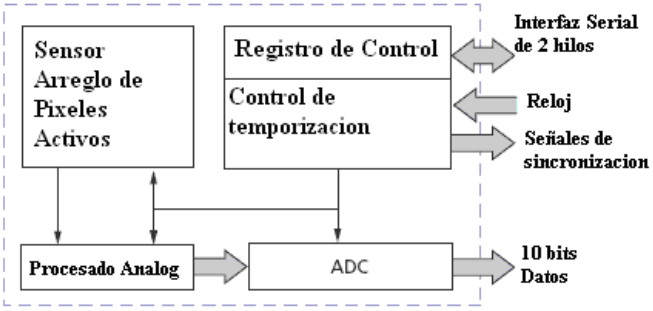


Figura 2.2 Diagrama a bloques del sensor MT9T001

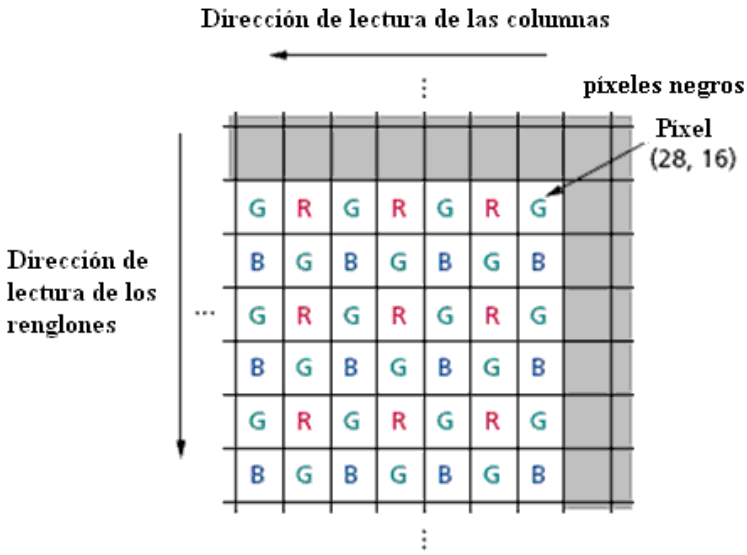


Figura 2.3 Arreglo de filtros de color con el patrón de Bayer

La imagen del MT9T001 es leída en un escaneo progresivo. Los datos válidos de la imagen están rodeados por un nivel de datos no válidos (blanking horizontal y blanking vertical). La cantidad de blanking vertical y horizontal es programable.

La señal LINE_VALID (línea válida) sincroniza la parte en la que se sacan los datos de los píxeles de una línea del cuadro de imagen. La señal FRAME_VALID (cuadro válido) sincroniza la parte en la que se sacan las líneas del cuadro de imagen. El dato de salida del MT9T001 esta sincronizado con la salida PXCLK. Cuando la señal LINE_VALID esta en alto, un dato de un píxel de 10 bits es sacado cada periodo del PXCLK. El dato DOUT es válido en el flanco de bajada de PXCLK en el modo predeterminado. El PXCLK depende directamente del reloj maestro. En la figura 2.4(a) se muestra un ejemplo de las señales de sincronización LINE_VALID y PXCLK y los datos de los píxeles, se observa que el PXCLK esta siendo continuamente habilitado, inclusive durante el periodo de blanking. En la figura 2.4 (b) se muestran las señales FRAME_VALID y LINE_VALID, los parámetros P, A y Q se definen según las ecuaciones especificadas en “Frame Timing Formulas” en [6]. Los tiempos del sensor pueden ser calculados usando dichas ecuaciones

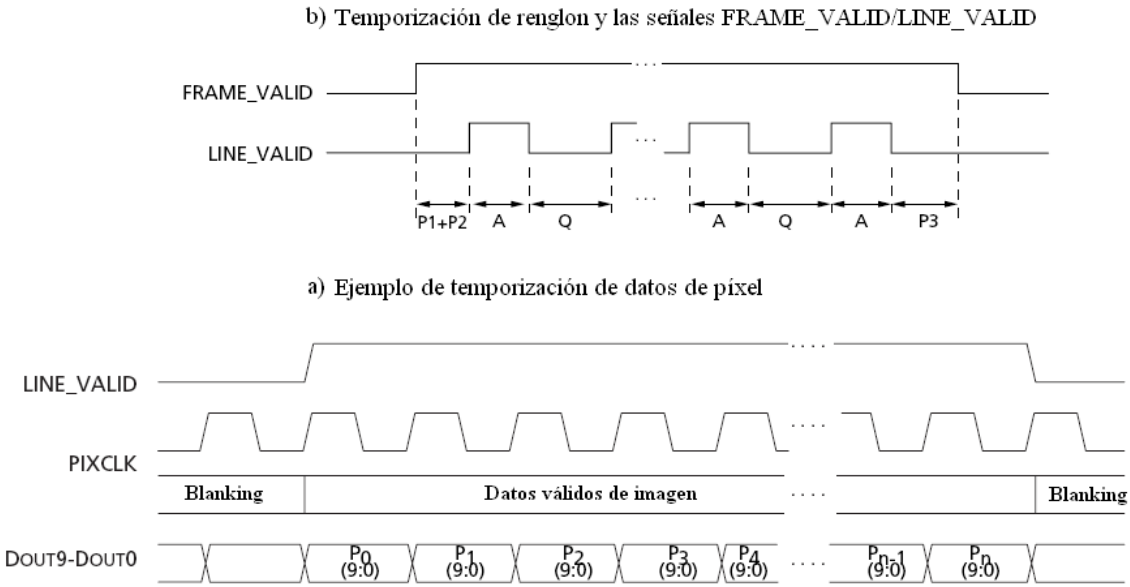
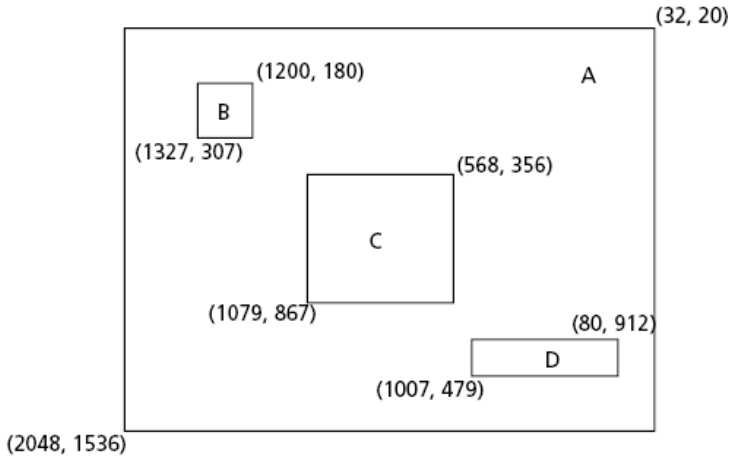


Figura 2.4 Señales de sincronización y datos del sensor MT9T001

El tamaño del arreglo de píxeles predeterminado es de 2,048 columnas por 1,536 renglones. Es posible modificar los valores de los registros internos de control del sensor MT9T001 para cambiar el tamaño de la ventana. Los registros Reg0x03 y Reg0x04 controlan la altura de la ventana (número de renglones-1) y el ancho (número de columnas-1). El tamaño de ventana más pequeño posible es 2 columnas por 2 renglones. El usuario puede programar el tamaño de la ventana para que sea cualquier formato deseado, en [6] se muestra ejemplos de ajustes para alcanzar varias resoluciones y tasas de cuadro

Además de cambiar el tamaño de la ventana, el usuario tiene la flexibilidad de cambiar la localidad de la misma. Los registros Reg0x01 y Reg0x02 controlan el primer renglón y la primera columna a ser leídos. La primera columna a ser leída debe ser un número par. Los registros Reg0x01 al Re0x04 permiten al usuario escoger cualquier segmento del arreglo del sensor para ser leído lo cual es especialmente útil cuando el usuario necesita hacer un acercamiento en una pequeña porción de la imagen y realizar análisis del contenido de la misma.



	Tamaño de Ventana	Reg0x01	Reg0x02	Reg0x03	Reg0x04
Ventana A	2,048 x 1,536	0x0014	0x0020	0x05FF	0x07FF
Ventana B	128 x 128	0x00B4	0x04B0	0x007F	0x007F
Ventana C	512 x 512	0x0164	0x0238	0x01FF	0x01FF
Ventana D	400 x 96	0x0390	0x0050	0x018F	0x005F

Figura 2.5 Ejemplos de colocación de ventanas programadas por usuario

La figura 2.5 muestra algunos ejemplos de la capacidad de ventaneo y colocación del sensor MT9T001. Los registros Reg0x05 y Reg0x06 controlan el tiempo que duran los datos no válidos en un renglón (blanking horizontal) y entre cuadros (blanking vertical), el blanking horizontal está especificado en términos de ciclos de reloj y blanking vertical en términos de tiempo de lectura de renglón.

El tiempo total de cuadro en términos de ciclos de reloj se puede obtener usando la fórmula dada en la tabla 2 de la página 11 de [6]. El usuario puede cambiar el número de columnas y renglones, tiempos de blanking horizontal y vertical para obtener diferentes tasas de cuadro.

Los registros son leídos y escritos al MT9T001 a través del bus de interfase serial de dos alambres. El MT9T001 es un dispositivo esclavo y es controlado por el reloj serial (SCLK), que es brindado por el dispositivo maestro. Los datos son ingresados y sacados del MT9T001 a través de la línea de dato serial (SDATA). La línea SDATA es puesta a voltaje de 3.3v fuera del chip (externamente) por una resistencia de 1.5kohms. Ya sea el dispositivo esclavo o el amo pueden poner a tierra la línea SDATA, el protocolo de comunicación serial determina cual dispositivo está permitido a poner a tierra la línea SDATA en un momento dado. La descripción de este protocolo se presenta en la sección 3.2.1.

2.1.3 Necesidad de la obtención de los 3 componentes de color a partir del arreglo de filtros de color (CFA)

Las cámaras digitales de color adquieren la información transmitiendo la imagen a través de filtros de color rojo, verde y azul, teniendo diferentes transmitancias espectrales y luego muestreando las imágenes resultantes usando 3 sensores electrónicos, usualmente CCD o CMOS. Para reducir el costo y la complejidad, los fabricantes de cámaras digitales usan un solo sensor CCD o CMOS con un arreglo de filtro de color (CFA) para capturar los tres colores primarios al mismo tiempo. [7]

Como se mencionó el sensor MT9T001 usa un arreglo de filtro de color con el patrón de Bayer donde cada píxel tiene un filtro de uno de los colores primarios por lo cual se necesita hacer un tratamiento para obtener los dos componentes de color faltantes de cada píxel a partir de los píxeles adyacentes. El patrón de Bayer es el más usado en las cámaras comerciales, existen otros patrones. En general a dichos patrones de color se les llama mosaicos y a los procesos para obtener los componentes de color faltantes, se les denomina interpolación de color o desmosaico ("desmosaicing").

Existen varios métodos para hacer el desmosaico, frecuentemente se les clasifican en dos tipos: adaptativos y no adaptativos. Los no adaptativos son aquellos algoritmos que realizan interpolación en un patrón fijo para cada píxel. Mientras los adaptativos pueden detectar características en la vecindad del píxel para tomar la decisión de cual patrón de interpolación usar para dicha vecindad.

Los algoritmos no adaptativos tienen la ventaja de tener el menor costo computacional lo cual es de vital importancia para el procesamiento de imagen en tiempo real. En [8] se tiene un estudio comparativo de varios métodos conocidos, se denota que la calidad de la imagen está en función de la complejidad computacional.

El algoritmo para el desmosaico más simple es la duplicación del vecino más cercano, el cual toma los colores faltantes de los píxeles más cercanos, este no tiene costo computacional ya que no realiza operaciones aritméticas pero tiene un desempeño bajo de calidad. Otro método de desmosaico muy utilizado es la interpolación bilineal debido a que tiene un bajo costo computacional con calidad de imagen aceptable, en la interpolación bilineal se calcula el valor de los dos componentes de color faltante de cada píxel haciendo el promedio de los píxeles vecinos, el proceso de interpolación bilineal se explica en la sección 3.2.3, la interpolación bilineal se aplica por medio de operaciones de vecindad que son similares en estructura a las operaciones de bajo nivel en imagen que se aplican por medio de la convolución.

2.2 Procesamiento de Imagen

2.2.1 Funciones de procesamiento de imagen de bajo nivel

Las operaciones de procesamiento de imagen de bajo nivel son operaciones matemáticas con estructuras bien definidas y que se aplican de forma repetitiva a todos los píxeles de una imagen de entrada dando como resultado una imagen de salida. En general para obtener la imagen de salida (imagen filtrada) se deben aplicar operaciones de vecindad, es decir para calcular cada píxel de salida se deben tomar en cuenta el píxel de entrada y los valores de los píxeles vecinos en la imagen de entrada. Comúnmente estas operaciones de vecindad trabajan con los valores de los píxeles en la vecindad y los valores correspondientes de una ventana que es de las mismas dimensiones que la vecindad, a los valores de la ventana se les llama coeficientes y a la ventana se le llama máscara del filtro.

Ejemplos de operaciones de bajo nivel en imagen son los filtros lineales, los cuales se aplican a través de la operación de convolución, en la cual el valor del píxel de salida es calculado a través de la suma de productos de los coeficientes de la máscara del filtro y los píxeles correspondientes de la imagen en el área abarcada por la máscara del filtro.

En la figura 2.6 se muestra un ejemplo de una operación de convolución con una máscara de 3x3, el píxel de salida se expresa con la siguiente ecuación: $h[i, j] = Ap_1 + Bp_2 + Cp_3 + Dp_4 + Ep_5 + Fp_6 + Gp_7 + Hp_8 + Ip_9$, se puede notar que los cálculos requeridos para obtener cada píxel de salida son nueve productos y ocho sumas, por lo que se entiende que este tipo de operaciones de bajo nivel demandan un gran cantidad de cálculos para imágenes de tamaños comúnmente usados, por lo que para cumplir las restricciones de procesamiento de imagen en tiempo real se deben utilizar aproximaciones novedosas para obtener las imágenes de salida.

También existen otras operaciones de bajo nivel que realizan cálculos en la vecindad de los píxeles para obtener el píxel de salida y no necesariamente realizan la convolución pero tiene estructuras parecidas.

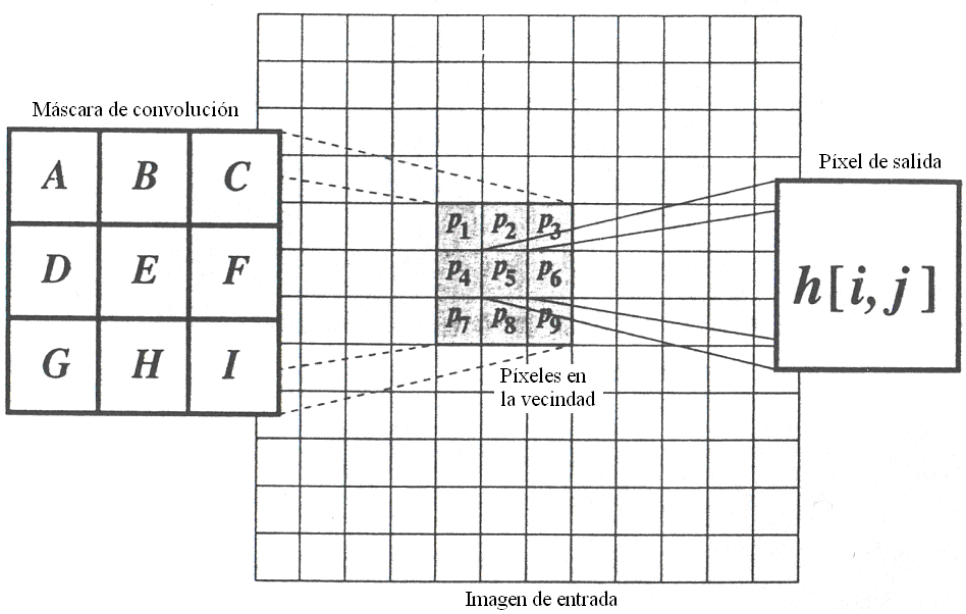


Figura 2.6 Ejemplo de un filtro lineal en imagen aplicado por medio de la operación de convolución

2.2.2 Uso de FPGA para procesamiento de imagen

En sistemas de visión en tiempo real que se utilizan en aplicaciones como inspección industrial, visión por máquina, control de trafico, sistemas de seguridad y en aplicaciones médicas, generalmente el tamaño de la imagen es grande y el tiempo de procesamiento debe ser pequeño, por lo que se debe tener especial cuidado en qué tipo de dispositivos se utilizarán como sensor y procesador de imagen.

El utilizar un procesador de propósito general para un sistema de visión en tiempo real no es una opción viable ya que las operaciones matemáticas repetitivas se llevarían a cabo de forma secuencial una a una por lo que el tiempo de procesamiento se incrementaría. Por otra parte el uso de hardware dedicado tal como los procesadores de señales digitales (DSP) ha sido la solución tradicional para estos sistemas sin embargo el desempeño de estos depende de cuantas operaciones en paralelo pueden realizar, en el caso de la convolución cuantas multiplicaciones podrían ser hechas en paralelo.

Los FPGA son una buena alternativa ya que al ser dispositivos reconfigurables permiten al usuario crear su propia arquitectura de procesamiento específica así por ejemplo para implementar las operaciones de bajo nivel en imagen se pueden usar una arquitectura que implemente tantos elementos procesadores en paralelo se requieran para las operaciones aritméticas necesarias.

Los sensores de imagen entregan los datos capturados píxel por píxel, renglón por renglón lo que resulta en una tasa de píxeles determinada. Para cumplir con las restricciones de tiempo real el procesador debe ser capaz de aceptar los datos de entrada a la frecuencia que el sensor de imagen los envíe, y de producir los datos de salida a la misma frecuencia de los datos de entrada. Para este fin lo más conveniente es usar una arquitectura en “pipeline”, en la cual básicamente se forma una “tubería” donde se almacenan las datos según van llegando y los elementos procesadores hacen los cálculos sobre aquellos datos que necesiten mientras siguen “fluyendo” en la tubería, de tal manera que no se debe esperar a que se realice un cálculo para empezar a realizar otro. En este esquema se tiene un tiempo de latencia inicial (el tiempo que dura en llenarse la tubería) y pasando este tiempo, se producen datos de salida la frecuencia de los datos de entrada.

Para moldear esas arquitecturas "pipeline" en un sistema digital se requiere de registros de corrimiento con el número de elementos adecuados según la aplicación. Las generaciones actuales de FPGAs cuentan con bloques de memoria RAM dedicados de tal forma que se pueden implementar los registros de corrimiento que formen las tuberías necesarias, además generalmente cuentan con bloques dedicados para implementar multiplicadores o en algunos dispositivos para implementar las operaciones de multiplicación acumulación. Por lo tanto son dispositivos bastante adecuados para realizar las operaciones de procesamiento de imagen en tiempo real.

Cabe mencionar que el tamaño de los registros de corrimientos requeridos (la cantidad de memoria) para implementar la mayoría de las funciones de bajo nivel en imagen equivale a tan solo algunas líneas del cuadro completo de imagen. En la figura 2.7 se muestra una arquitectura genérica para realizar el filtro de convolución en tubería, se puede observar que los buffer de línea son registros de corrimiento en los que se almacenan los píxeles de una línea del cuadro de imagen y que el número de estos depende del tamaño de la ventana de convolución.

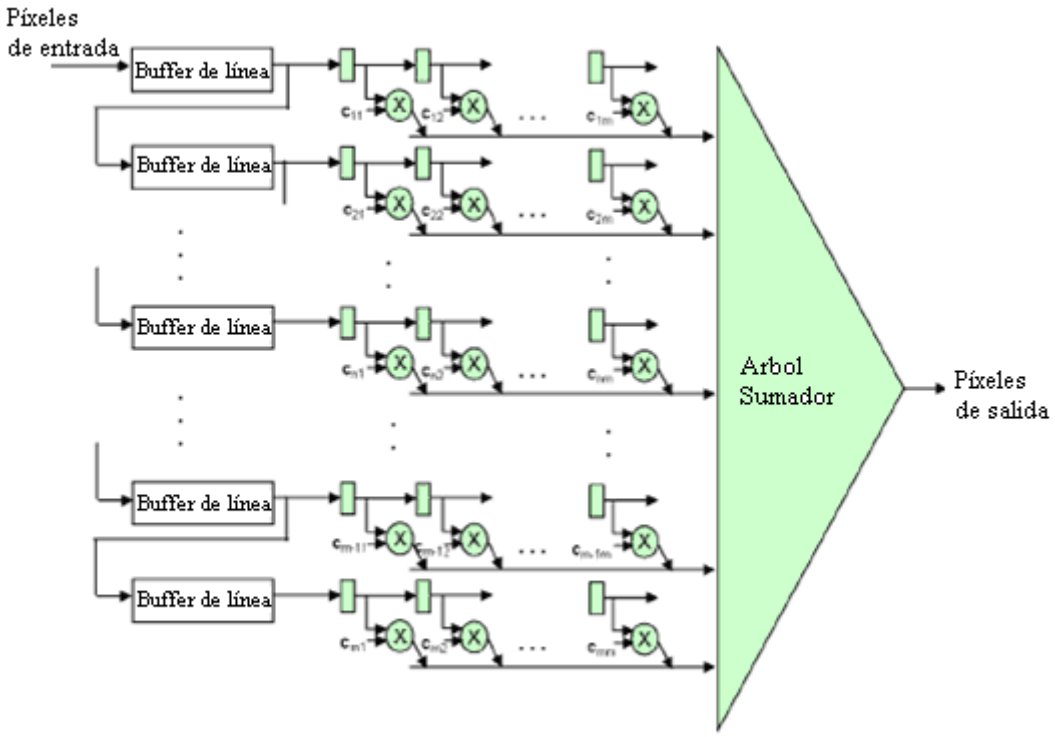


Figura 2.7 Arquitectura genérica para implementar el filtro de convolución en imagen, con una ventana

2.3 FPGA

2.3.1 Historia de la lógica programable

Los dispositivos lógicos programables (PLD por sus siglas en inglés) se desarrollaron con el fin de sustituir los circuitos de baja y mediana escala de integración (SSI y MSI) tradicionales tales como los de lógica de transistor transistor (TTL), en los cuales se tienen compuertas lógicas fijas y se deben de alambrar un conjuntos de circuitos para realizar funciones básicas como sumas de productos. La arquitectura básica de un PLD esta formada por un arreglo e compuertas AND y OR conectadas a las entradas y salidas del dispositivo.

Los dispositivos de memoria programable de solo lectura (PROM) constan de compuertas AND fijas que trabajan como decodificadores y compuertas OR programables, de tal forma que se usa como una memoria direccionable también, permiten implementar circuitos combinacionales en formas de tablas de búsqueda (LUT).

El primer PLD desarrollado para implementar circuitos lógicos fue el arreglo lógico programable (PLA) (a mediados de los 70s) que surgió para superar las limitaciones del PROM debido al arreglo AND fijo, los PLA cuentan con un arreglo de compuertas AND de entrada y un arreglo OR de salida ambos programables, usan interconexiones en forma de barras cruzadas o rendijas donde se usa la tecnología de transistores bipolares y fusibles (Bipolar fuse link technology) (para unir las entradas al arreglo AND y las salidas de las ANDs a las entradas de las OR por medio de un fusible para cada entrada y salidas), donde cada salida se conecta a cada entrada a través de un sólo switch programable (fusible), los PLA permiten realizar operaciones combinacionales como sumas de productos en dos niveles (cada salida es la suma de los términos de productos seleccionados). Estos tuvieron un éxito limitado ya que eran algo lentos y difíciles de usar [9].

Los dispositivos PAL (lógica de Arreglo programable) surgieron a finales de los 70s, los PAL tiene un arreglo de compuertas AND programable de entrada y un arreglo fijo de salida de compuertas OR, su arquitectura establece que cada salida es la suma de un juego especifico de productos de términos, también se incluyeron flip-flops en las salidas de las compuertas OR para poder implementar circuitos lógicos secuenciales.

El PAL se desarrolló para superar algunas limitaciones del PLA, como retardos provocados por la implementación de fusibles adicionales, que resultan de la utilización de dos arreglos programables y de la complejidad del circuito [10].

Los dispositivos GAL (Arreglo Genérico lógico) son similares a los PAL ya que constan de un arreglo AND programable y uno OR fijo pero los GAL son reprogramables además constan de elementos llamados macroceldas lógicas de salida (OLCM Output Logic Macrocell) dichos elementos están constituidos por circuitos lógicos y flip-flops que pueden ser programados para implementar lógica combinacional o secuencial es decir; cuenta con configuraciones de salida programables, cabe mencionar que los GAL están fabricados con tecnología EECMOS (CMOS eléctricamente borrable) por lo cual se pueden reprogramar cuantas veces sea necesario en lugar de usar tecnología bipolar y fusibles.

Los dispositivos lógicos programables (PLDs) se constituyen en un solo circuito integrado; son dispositivos fabricados y revisados que se pueden personalizar desde el exterior mediante diversas técnicas de programación. El diseño se basa en bibliotecas y mecanismos específicos de mapeo de funciones, mientras su implementación tan solo requiere una fase de programación del dispositivo, que por lo general realiza el diseñador en un tiempo corto. [10]

2.3.2 PLD de alto nivel de integración o FPLDs

Los FPLDs[4] representan un desarrollo relativamente nuevo en el área de los circuitos integrados de muy alto nivel de integración (VLSI), implementan miles de compuertas lógicas en estructuras de múltiples niveles [11]. Los FPLDs se componen de múltiples PLD sencillos y cuentan con arquitectura de interconexión con ruteos más eficientes donde usualmente cada conexión pasa a través de varios switches.

Los FPLDs o PLDs de alto nivel de integración se desarrollaron con el fin de integrar un mayor número de componentes en un solo chip con lo que se logra la reducción de espacio, costo y permiten implementar sistemas digitales complejos y versátiles ya que la velocidad y frecuencias de operación se han incrementado con respecto a los PLD sencillos, además habilitan a los diseñadores a ingresar productos al mercado mas rápidamente y hacer modificaciones en el diseño sin afectar al dispositivo.

2.3.3 Dispositivos Lógicos Programables Complejos (CPLD)

Los dispositivos lógicos programables complejos (CPLDs por sus siglas en inglés) se componen de **bloques lógicos** los cuales se comunican por medio de un matriz o arreglo de interconexión programable la cual se encarga de unir bloques lógicos con las **celdas de entrada/salida.** Los bloques lógicos se constituyen por arreglo de productos de términos para implementar los productos de las compuertas AND, por un **esquema de distribución de términos** que habilita la canalización de dichos productos hacia sumas deseadas y por macroceldas que internamente se componen de flip-flops;

[4] En la literatura se consideran dispositivos lógicos programables en campo (FPLD) a los PLD de alto nivel de integración es decir, a los dispositivos lógico programables complejos (CPLD) y a los arreglos de compuertas programables en campo (FPGA), el termino “en campo” se refiere a que se programan en el campo de trabajo

El número de macroceldas por bloque lógico se refiere al número de registros básicos disponibles para implementar funciones lógicas secuenciales tales como máquinas de estado por lo que usualmente los bloques lógicos definen su capacidad en términos del número de macroceldas que contienen. Otro parámetro importante es el número de entradas disponibles hacia los bloques lógicos desde la matriz de interconexión ya que si un diseño requiere mas entradas de las disponibles se tendrá que usar otro bloque lógico para implementarla. También el número de productos de términos que el dispositivo puede sumar juntos en un bloque lógico es un parámetro que limita el monto de lógica combinacional que puede caber en el dispositivo.

2.3.4 Arreglo de Compuertas Programable en Campo (FPGA)

Estos dispositivos se basan en lo que se conocen como arreglos de compuertas y provienen de los llamados arreglos de compuertas programables por máscara[5] (MPGA). Los FPGA se conforman internamente de **bloques lógicos configurables** (CLB) que se comunican con los **bloques de entrada/salida** (IOB) por medio de **canales de interconexión.** Cada FPGA cuenta con un arreglo de bloques lógicos idénticos, este arreglo está dispuesto generalmente en forma de matriz cuadrada, donde los bloques lógicos se comunican entre si por medio de líneas metálicas horizontales y verticales (canalizadas por medio de la red o matriz de interconexión programable). Los bloques lógicos tienen la capacidad de implementar cualquier función booleana en forma de productos de suma y funciones lógicas secuenciales.

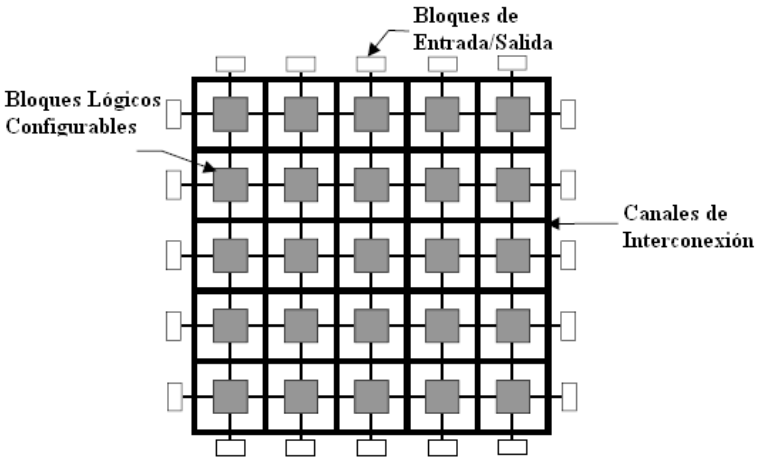


Figura 2.8 Estructura general de un FPGA

[5] Los MPGA son circuitos integrados que consisten de un arreglo de transistores prefabricados que se requieren personalizar durante la fabricación del chip

Los bloques lógicos están basados en uno o más de los siguientes elementos lógicos como bloques constructores: pares de transistores, compuertas pequeñas, tales como NANDs o XORs de dos entradas, multiplexores, tablas de búsqueda (LUT); estructuras AND-OR de muchas entradas [9].

En FPGA modernos lo más común es encontrar que cada bloques lógico configurable se constituye por un arreglo de elementos lógicos que se basan en tablas de búsqueda (LUT Look Up Table) es decir; que la lógica combinacional no se implementa usando compuertas lógicas sino que al configurar el dispositivo (programarlo) se carga la lógica a una memoria SRAM de alta velocidad que actúa como una tabla de verdad de tal forma que una combinación de entradas brinda la dirección de la memoria y el dato almacenado es la salida que se alimenta a un flip-flop y después a la red de interconexión.

El tamaño y flexibilidad (granularidad) de los elementos lógicos se pueden definir en varias formas, tales como el número de funciones booleanas que se pueden implementar, el número de compuertas NANDs de dos entradas equivalentes, el número total de transistores o el número de entradas/salidas. [9]

Los bloques de Entrada/Salida (E/S) se encuentran localizados en cada una de las terminales de E/S del dispositivo, generalmente contienen un registro de un bit (un flip-flop) con un manejador de tres estados (tri-state driver); de tal forma que cada pin de E/S se pueda configurar como entrada, salida, salida con manejador de tres estados o incluso como bidireccional de tres estados con o sin registro. Los bloques lógicos configurables pueden conectarse a la matriz de interconexión programable o directamente a los bloques de E/S.

La capacidad (tamaño) de un FPGA se define usualmente por la cantidad equivalente de compuertas lógicas disponibles en el dispositivo. Esto se refiere al máximo número de compuertas NAND de dos entradas disponibles en el dispositivo. Esto debe ser visto como una estimación ruda del tamaño solamente, la utilización actual de compuertas en un diseño particular puede variar significativamente [12]

Los FPGA y CPLD no tienen una brecha nítida que los separe sin embargo, tradicionalmente la diferencia principal ha sido que los FPGA cuentan con mayor número de registros (flip-flops) mientras que los CPLDs con su arquitectura mas parecida a los PLD sencillos es más apto para diseños de lógica combinacional. Por otra parte los CPLD tienden a tener propiedades de tiempo más rápidas y más predecibles por lo que pueden alcanzar mayores frecuencias de trabajo que los FPGA. [12]. Además cabe mencionar que los FPGA modernos cuentan con bloques dedicados de memoria RAM, bloques con multiplicadores embebidos e incluso bloques dedicados que cuentan con las estructuras para aplicar las operaciones de multiplicación acumulación (MAC), los que los hace excelentes en aplicaciones de arquitectura de computadoras, procesadores digitales de señales y diseños con registros.

En cuanto a las frecuencias de operación los FPLDs manejan tasas de velocidad adecuadas para aplicaciones con restricciones en tiempo real con tasas máximas de reloj en el rango de 50 a 400Mhz; algunos dispositivos han alcanzado tasas de reloj de hasta 1Ghz. Las señales de reloj en los FPLD utilizan líneas especiales con buses de alta velocidad para que todos los flip-flops sean alimentados al mismo tiempo para minimizar el "clock skew"[6]

Los ASIC (Circuitos Integrados para Aplicación Especifica) son circuitos que se desarrollan para una aplicación en particular y que requieren personalizarse durante la fabricación del chip, usualmente el usuario escoge diseños de una biblioteca del fabricante para implementar circuitos de alto nivel de integración como memorias RAM, ALU, microprocesadores, etc. Los ASIC presentan la mayor eficiencia en cuanto espacio ocupado por el diseño y la velocidad o frecuencias de operación ya que no tienen red de interconexión programable sino que físicamente se interconectan las celdas internas por lo cual los retardos debidos a alambrado son mínimos.

[6] El Clock skew se refiere a los circuitos síncronos en los cuales la señal de reloj llega a diferentes componentes en tiempos diferentes

La desventaja es que los ASIC son circuitos que una vez fabricados no se pueden modificar y debido a que requieren fabricación personalizada típicamente se necesitan de varios meses de ingeniería para tener el producto terminado además de los costos adicionales que se involucran por la fabricación. Esto hace que los ASICs sean adecuados solamente para productos con largo tiempo de vida y en grandes volúmenes de producción ya que el costo por unidad se abarata en comparación con los FPGAs y CPLDs; sin embargo los costos iniciales de ingeniería para los ASICs son más altos

Los FPGA son dispositivos adecuados para crear prototipos de forma rápida y a costo barato, en este momento la investigación que se lleva en sistemas electrónicos digitales se ha vuelto más accesible gracias al uso de los FPLDs ya que las herramientas utilizadas para configurarlos se han difundido ampliamente en los años recientes y los costos de los dispositivos de este tipo son relativamente baratos además de que en el ámbito académico los fabricantes ofrecen precios más accesibles. Los FPGAs permiten implementar una amplia gama de sistemas digitales y habilitan al diseñador a crear prototipos en cuestión de días o semanas, probarlos y reconfigurarlos en caso de haber errores o si se desea mejorar el diseño existente, además los FPGA permiten realizar un diseño desde el principio (“from de scratch”) es decir; crear todo la lógica necesaria, haciendo el código en un lenguaje de descripción de hardware que pueden ser trasladado a los FPLDs Y ASCIs.

2.4 VHDL

2.4.1 Herramientas de diseño de la lógica programable

Para el diseño con PLD se desarrollaron herramientas de CAD en las cuales la forma tradicional de ingresar los diseños se basa en la representación gráfica (esquemática) del circuito que se quiere implementar y partiendo de éste, se utiliza algún método adecuado como ecuaciones booleanas, tablas de verdad o diagramas de estado para solucionar el modelo e implementarlo al PLD; esto se sigue usando para PLD de baja densidad sin embargo, no es práctico para realizar diseños complejos en dispositivos FPLDs debido al alto número de elementos lógicos con los que cuentan y la complejidad de la red de interconexión programable.

Los FPLD requieren de un ambiente uniforme de diseño (software CAD) que integre varias herramientas con funciones específicas (o pasos del mismo diseño), lo que habilite al usuario a trabajar a distintos niveles de abstracción, por ejemplo: ingresar el diseño y dejar a la herramienta de mapeo (fitting) decidir qué elementos del dispositivo usar para la implementación sin que el usuario lo indique o incluso si es necesario que sea posible ir al más bajo nivel de abstracción para hacer este tipo de tareas.

El ambiente de diseño debe de ser independiente de la arquitectura del FPLD y debe conformarse de algunas herramientas primarias que realicen el **ingreso del diseño en** algún formato estándar (esquemático, lenguaje HDL, editores de forma de onda, etc.), la **traducción** de dicho formato para las arquitecturas de los FPLD, donde el traductor debe realizar varias funciones como síntesis lógica[7], compilación por tiempos, y la colocación o mapeo (fitting) del diseño en la arquitectura particular del FPLD, la **verificación del diseño** usando simulación funcional y de tiempos que permite encontrar los errores en el diseño antes de programar el dispositivo, la **programación del dispositivo** que consiste en cargar a la información del diseño al FPLD, y la **reutilización** que se refiere a que el ambiente debe permitir reutilizar unidades de diseño (en cualquier formato de ingreso) hechas por el usuario o provenientes de bibliotecas del fabricante.

[7] La síntesis lógica en general, es el proceso de generar un netlist (Un netlist es una representación de un diagrama lógico basada en texto) optimizado a nivel compuertas lógicas para un diseño lógico a partir de una descripción estructural de alto nivel [9].

Usualmente los dos formatos mas utilizados para ingresar diseños son los de tipo esquemático y textuales (lenguajes HDL). Los ambientes de diseño típicamente permiten utilizar combinaciones de ambos formatos

2.4.2 Lenguajes de descripción de hardware.

Los lenguajes de descripción de hardware (HDL) o de alto nivel de descripción permiten ingresar los diseños en forma textual (parecida a los lenguajes de programación de alto nivel), soportan la creación de sistemas electrónicos digitales complejos a varios niveles de abstracción, se usan para describir el comportamiento, estructuras de diseño y sus interconexiones, los HDL son modulares ya que los módulos (subsistemas, proyectos) creados usando HDL se pueden integrar dentro de un diseño(proyecto) más grande, a esto se le conoce como metodología de diseño jerárquico, donde el sistema mas grande se le llama diseño de alto nivel (Top Level Design).

El diseño jerárquico tiene varias ventajas tales como, facilidad de depuración, partición de diseños grandes y complejos, almacenamiento de funciones comúnmente usadas en una biblioteca de macros para su repetido uso. [9]

Una descripción hecha en un lenguaje HDL puede ser implementada en dispositivos con distintas tecnologías (CPLD, FPGA y ASIC), ya que las herramientas de síntesis lógica se encargan de mapear las funciones a las diferentes arquitecturas sin embargo, no es una tarea fácil ya que generalmente se debe de seguir un cierto estilo de código para que el diseño se adapte mejor a una arquitectura particular.

Los niveles de abstracción usados en un código HDL dictan el nivel de detalle de la descripción de cierto sistema digital, se pueden definir 3 niveles según las herramientas de síntesis los manejan. Nivel a**lgoritmo:** se refiere a la relación entra las entradas y salidas del sistema, esta es una descripción a nivel funcional (no detalla que la interconexión de compuertas lógicas). Nivel **transferencia de registros (RTL)**: consiste en la partición del sistema en bloques funcionales sin considerar a detalle la constitución de cada uno. Nivel **lógico o de compuertas**: el circuito se expresa en términos de ecuaciones lógicas o de compuerta. Los HDL pueden ser lenguajes diseñados por los fabricantes o lenguajes estandarizados que son independientes de los vendedores.

Como resultado de las necesidades y desarrollos de las metodologías para diseño de sistemas digitales, emergieron VHDL y Verilog como herramientas estándar para la descripción de sistemas digitales en varios niveles de abstracción optimizados para transportabilidad ante muchos ambientes de diseño de computadora.[11]

2.4.3 VHDL (Very High Speed Integrated Circuit Hardware Description Language)

En los años 80's el departamento de defensa de E.U.A se vió en la necesidad de crear un medio estándar para describir, modelar y documentar todos los diseños de sus ASICs, así surgió VHDL como parte del proyecto VHSIC, que se puede entender como el afán de diseñar circuitos integrados de la forma más rápida posible. Después de varias versiones de VHDL revisadas por el gobierno de E.U.A, industrias y universidades, la IEEE definió VHDL por medio del estándar IEEEstd 1076-1987 y la revisión 1076-1993 además se han continuado con actualizaciones constantes, mejoras y metodologías de uso. Actualmente VHDL es el lenguaje HDL más usado en la industria. [10]

VDHL se considera un estándar para la descripción, modelado y síntesis de sistemas digitales y ha tenido un enorme impacto en la metodología ya que promueve el proceso de diseño jerárquico. VHDL es un lenguaje que sigue la filosofía de reutilización de código y a diferencia de los lenguajes de programación convencionales[8] está diseñado para modelar operaciones en paralelo (operaciones concurrentes) lo cual es importante ya que los sistemas digitales hardware operan en paralelo por su propia naturaleza. VHDL permite ingresar un diseño a distintos niveles de abstracción esto es, se puede describir un sistema de forma algorítmica (funcional) o si se requiere hacerlo hasta a nivel de compuertas lógicas e incluso combinar porciones del código en varios niveles dentro de un mismo diseño. La metodología de diseño jerárquico hace legibles y comprensibles los códigos ya que se puede entender el funcionamiento general del diseño sin adentrarse a los detalles de bajo nivel. VHDL facilita el trabajo en equipo ya que se pueden dividir los proyectos en los componentes que los conforman y dejar la descripción de cada componente a un integrante del equipo.

[8] Los lenguajes programación convencionales no pueden modelar adecuadamente los procesos en paralelo debido a que se basan en un modelo secuencial de operación

VHDL es un estándar no sometido a patente o marca registrada alguna, por lo que cualquier empresa o institución puede utilizarla sin restricciones. Ya que es un lenguaje estándar los códigos hechos en VHDL pueden implementarse en distintas tecnología de dispositivos y transportarse a diferentes ambientes de diseño CAD por lo que es común que los códigos se manejen como propiedad intelectual de los fabricantes o desarrolladores que los comercializan ofreciendo bibliotecas (funciones) de sistemas digitales para que los usuarios escogen (compren) según sus requerimientos. Cabe mencionar que cualquier el usuario/investigador puede crear sus propios códigos para sus aplicaciones.

El lenguaje VHDL consiste de varias partes: EL lenguaje como tal especificado en el estándar de la IEEE. Algunas declaraciones adicionales de tipos de datos en el paquete llamado estándar IEEE 1164. Una biblioteca llamada WORK usada para los diseños del usuario. Además que el usuario puede crear sus propios paquetes o bibliotecas así como adquirir los mismos de los vendedores.

El diseño en VHDL consiste de varias unidades separadas, cada una es compilada y guardada en una biblioteca. Las cuatro unidades de diseño que pueden ser compiladas son entidad, arquitectura, configuración y paquete, a continuación se explican brevemente.

Entidad es la unidad más básica, describe el tipo y dirección de las señales de entrada/salida del diseño, una vez que se compila una entidad se puede simular o se puede usar como componente de otro diseño mayor, en VHDL se separa la descripción de la interfaz E/S de descripción del funcionamiento (arquitectura) de la implementación. Todos los diseños son creados a partir de las entidades, una entidad en VHDL corresponde directamente a un bloque en la metodología tradicional de ingreso esquemático. Adicionalmente se puede usar un tipo de listado especial llamado genérico (generic) que permite ingresar parámetros que serán usados en el diseño como por ejemplo número de bits de los puertos, número de registros en una tubería, etc.

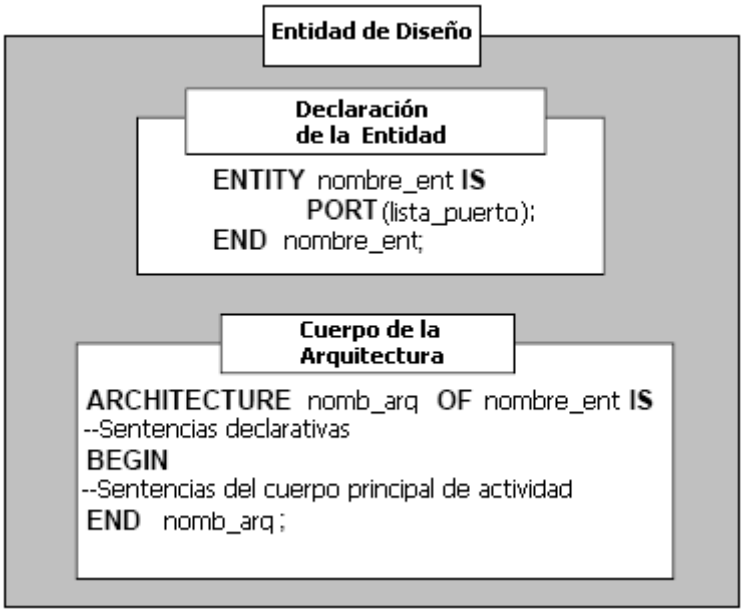


Figura 2.9 Entidad de diseño VHDL

Arquitectura. Describe el comportamiento, interconexiones y componentes de una entidad. Las relaciones entre las entradas y las salidas se pueden expresar en términos de los estilos de **comportamiento**, **flujos de datos** o **estructura**l. La arquitectura en estilo de **comportamiento** describe el comportamiento de una manera algorítmica o "parecida a un lenguaje convencional". Cuando se usa diseño jerárquico se usan entidades previamente compiladas y se integran en el diseño de alto nivel usando el estilo **estructural** que es un listado de las interconexiones de los componentes (entidades previamente compiladas). Arquitectura en estilo de **flujo de datos** modela la información o el comportamiento del flujo de datos de funciones lógicas combinacionales, como sumadores, comparadores, multiplexores y otros circuitos lógicos primitivos. Una entidad puede implementar más de una arquitectura, se selecciona por medio de la configuración.

La **configuración** se usa para especificar qué arquitecturas usar para cada entidad, ayuda al diseñador a experimentar con diferentes variaciones de diseño seleccionando arquitecturas particulares. Se usa para ligar un componente a un par entidad arquitectura

Un **paquete** almacena especificaciones usadas frecuentemente tales como tipos de datos, declaraciones de componentes, declaraciones de constantes entre otros, se pueden considerar como una caja de herramientas usada para construir diseños. Los elementos definidos en un paquete se pueden utilizar por cualquier entidad

Biblioteca almacenan los resultados de las compilaciones de todas las unidades de diseño para la simulación subsiguiente o para su uso en otros proyectos (diseños), la biblioteca puede contener paquetes (declaraciones compartidas), entidades, arquitecturas, configuraciones Existen dos bibliotecas integradas (por default) (WORK y STANDARD), pero el usuario puede crear otras bibliotecas. Las unidades en la biblioteca deben tener nombres únicos; todos los nombres de las entidades de diseño y los nombres de paquetes deben ser únicos dentro de una biblioteca. Los nombres de las arquitecturas necesitan ser únicos para un diseño en particular.

Capítulo III

Metodología

3.1 Búsqueda del entorno de desarrollo

3.1.1 Ambiente de trabajo para desarrollo de un sistema digital

El objetivo principal de este proyecto de cámara inteligente, se traduce a un sistema compuesto por un sensor de imagen CMOS que enviara sus datos capturados directamente a un FPGA, en donde se diseñaría una arquitectura de procesamiento para probar las ventajas y flexibilidad del uso de este dispositivo y procesar los datos a la misma tasa que entran. El logro de este objetivo nos exige contar con un ambiente de trabajo con las herramientas hardware y software necesarias, para poder explorar el diseño de la arquitectura.

Para iniciar las pruebas o experimentos del proyecto, primero se adquirió el hardware y software que se consideraban lo mínimo necesario y se revisó cual ya se tenía a la mano y en que podría ser útil. Cabe mencionar que se desarrolló un ambiente de trabajo propio, es decir, no se compró algún kit de desarrollo donde ya se tuviera el sensor de imagen conectado a un FPGA. El Hardware que se utilizó fue el siguiente:

- Una tarjeta MI3100 de Micron que trae el sensor de imagen CMOS MT9T001, circuito de alimentación, conectores y juego de lentes.
- También se adquirió la tarjeta DEMO2 que es un sistema que se conecta a la tarjeta MI3100 y a otras tarjetas del fabricante Micron, para evaluar los sensores de Imagen.
- Tarjetas de desarrollo UP2 que trae un FPGA FLEX10K y la DE2 con FPGA Cyclone II ambas de Altera.
- Tarjetas de entradas/salidas digitales de National Instruments (NI), DAQPAD6507 y la PCI-6023E.

- Para las mediciones se contaba con un Osciloscopio analógico de 4 canales Tektronix TDS460 y se consiguió un osciloscopio de señal mixta Agilent 54621D con 16 entradas digitales y dos analógicas.

- Por otra parte, el software que se utilizó fue el siguiente:

- Se consiguió la versión libre (versión web) del software QUARTUS II, que es una herramienta CAD para desarrollar aplicaciones para CPLDs y FPGAs de la compañía Altera.

- También se contaba con el software Matlab de MathWorks, el cual es un ambiente de desarrollo que cuenta con el DAQ Toolbox, el cual es una herramienta para adquisición de datos que permite conectar dispositivos externos de adquisición de distintos fabricantes (entre ellos, los dispositivos de NI).

- También se consiguió el software manejador NI-DAQ, que es una biblioteca de funciones para los dispositivos de NI que se mandan llamar desde un software de aplicación como C++, Matlab, etc.

- Se adquirió el software DevWare, el cual viene con el sistema DEMO2 de Micron y se utiliza para evaluar sensores de imagen de Micron, permite visualizar imagen del sensor, escribir a sus registros internos entre otras funciones.

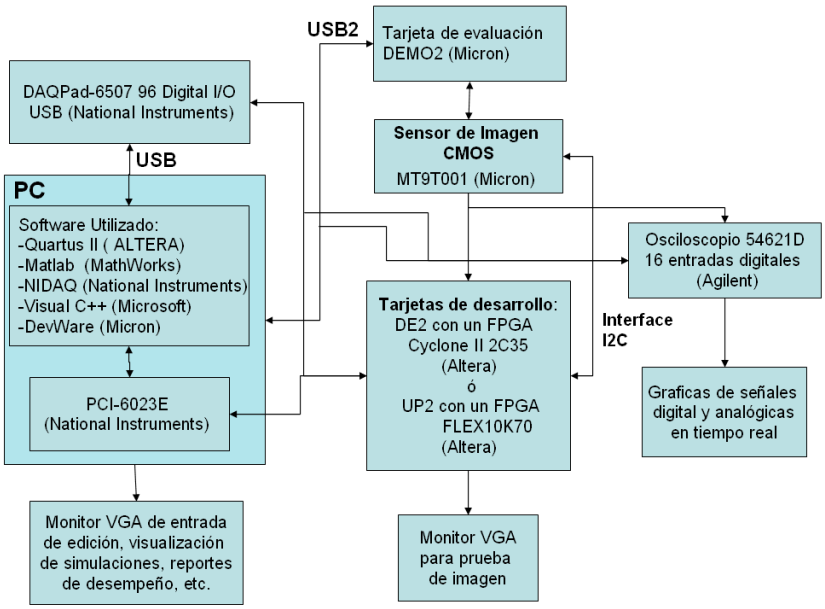


Figura 3.1 Entorno de desarrollo, mostrando los elementos hardware y software que lo constituyen

3.1.2 Metodología de diseño lógico programable (metodología de diseño para FPGA utilizando VHDL)

La metodología de diseño para un dispositivo FPGA utilizando el lenguaje VHDL, requiere de un ambiente de desarrollo CAD conformado por varias herramientas: que permitan ingresar el diseño, traducirlo para la arquitectura específica del dispositivo FPGA, simular su comportamiento de forma funcional o por tiempos y cargar o configurar el dispositivo. VHDL permite desarrollar la metodología de diseño jerárquico para dividir el diseño en cada uno de las componentes (o módulos) que lo conforman, pudiendo simularlos y verificarlos por separado y después unirlos en un diseño de alto nivel. Además, brinda la oportunidad de reutilizar los componentes que son comunes para diferentes diseños mayores. En esta tesis se utilizó el software CAD QUARTUSII (de Altera) que cumple con todos los requerimiento mencionados.

La metodología de diseño digital usada en esta tesis se puede describir brevemente de la manera siguiente:

a) Ingresar la descripción del sistema digital utilizando el lenguaje VHDL.

b) Simular el comportamiento del diseño usando archivo con las formas de onda (diagramas de tiempos) de las señales de entrada/salida y registros, de tal forma que se puede editar las señales de entrada manualmente o pueden provenir de otro diseño ya existente.

c) Verificar.- Si hay errores en la simulación se corregía el código VHDL.

d) Cargar el diseño ya simulado al FPGA y probarlo en el ambiente físico.

Dentro de este contexto de diseño jerárquico, se empezaron a definir los componentes[9] que se requerían para implementar el sistema de cámara inteligente en el FPGA. Además de definir los elementos externos al FPGA necesarios para el desarrollo del proyecto (explorar el ambiente de trabajo)

[9] En esta tesis "componentes" son las descripciones hechas en VHDL de los sistemas digitales desarrollados para el dispositivo FPGA

3.1.3 Ejercitar el FPGA para manejar las señales de sincronización y datos del sensor MT9T001

Se realizaron códigos en VHDL para que el FPGA aceptara las señales de sincronización y datos del sensor MT9T001 (dichas señales se exponen en 2.1.2) y así poder manejarlas para su posterior procesamiento.

Siguiendo la metodología mencionada en 3.1.3, se hizo un componente llamado SERIAL.vhd[10] que acepta las señales de sincronización obteniendo cada dato en el flanco de bajada del reloj y mandándolo a la salida un ciclo de reloj después. El componente RAM.vhd crea un bloque de memoria, en el cual en cada flanco de bajada del reloj se guarda dato a localidad de memoria, mientras que en el flanco de subida se saca el dato al igual que actualiza la localidad, de tal forma que al llegar a la última localidad se volvía a la primera continuando el proceso.

Se simularon dichos componentes, ingresando manualmente las señales de entrada (sincronización y datos) en el archivo de forma de onda y verificando el comportamiento de las señales de salida. Una vez que se depuraron los errores y se obtuvieron simulaciones adecuadas se pudo hacer la verificación en la práctica.

Para verificar este componente en la práctica se necesitaba conectar las señales del sensor MT9T001 al FPGA sin embargo, se consideró que sería mas adecuado primero tener un ambiente controlado para adquisición y verificación de datos, por lo que se decidió generar señales que simularan las del sensor por medio de la PC.

[10]En el apéndice A se muestran los códigos en VHDL, programas en Matlab y en Visual C++ , realizados de la sección 3.1.1 a 3.1.11

3.1.4 Generar las señales que simulan las del sensor

Como ya se mencionó en 3.1.1, Matlab cuenta con una herramienta para adquisición de datos ("Data Acquisition Toolbox"), esta es un software de aplicación que se enlaza con algún software manejador ("driver software") que permite interactuar con dispositivos hardware externos de distintos fabricantes. Se utilizó el software manejador NI-DAQ y la tarjeta PCI-6023E de NI

Se creó el programa Prueba.m (con funciones de Matlab y del DAQToolbox) para generar las señales de sincronización y datos que simulan las del sensor de imagen CMOS, en dicho programa los datos se envían siguiendo las señales de sincronización y provienen de una matriz de datos que se puede editar en el mismo.

Se verificó el funcionamiento del sistema conectando las salidas de la tarjeta PCI-6023E a las entradas correspondientes del FPGA FLEX10K y se midieron las señales en el osciloscopio. Se probaron los componentes SERIAL.vhd y RAM.vhd y se editaron distintos valores de la matriz de datos que se enviaba desde Matlab por medio del programa Prueba.m. En la figura 3.2 se observan las señales de sincronización que simulas las del sensor MT9T001 generadas.

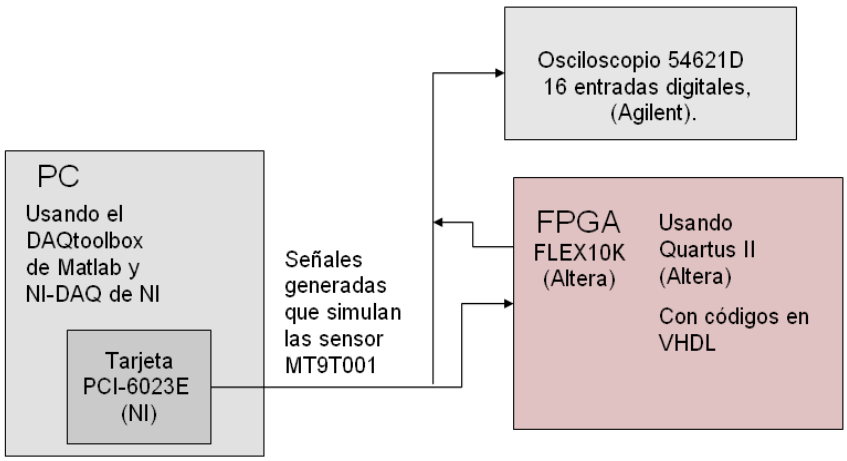


Figura 3.2 Diagrama a bloques de los dispositivos utilizados para generar y aceptar las señales que simulan las del sensor MT9T001

3.1.5 Evaluación de la capacidad de Matlab para transferencia de datos utilizando los dispositivos de NI

Se propuso enviar una matriz de datos desde Matlab (DAQToolbox) por medio de la tarjeta DAQPad-6507 al FPGA FLEX10K70, guardar la matriz de datos y reenviarla a la tarjeta NI con lo que Matlab pudiera retomar dicha matriz. El primer paso fue saber si Matlab podría adquirir los datos provenientes del FPGA, se encontró con el problema de que el DAQToolbox de Matlab no cuenta con funciones para detección de flancos, se hizo un programa en Matlab que intento hacer la detección de flanco usando la técnica de programación por sondeo ("polling").

Se hizo el componente ENVIAR.vhd el cual por medio de una señal de aviso (de un pushbutton) envía datos desde un arreglo de memoria interno con valores fijos (ROM) y divide la frecuencia del reloj de la tarjeta UP2 (25Mhz) para obtener una señal de reloj que sincroniza los datos enviados y por lo tanto se pueda manejar la tasa de envío. Se verificó el componente ENVIAR.vhd por medio de las mediciones del osciloscopio.

Se procedió a probar el programa de Matlab (por sondeo) que solo recibiera los datos del FPGA pero no funcionó, se observó que no había control de la transferencia sin la detección de flancos, se encontró en [13] que las funciones del DAQtoolbox de Matlab no soportan handshanking[11] para dispositivos de entradas/salidas digitales.

Se encontró que las funciones de NI-DAQ sí permiten handshaking con dispositivos de E/S digitales. Se adquirió una biblioteca (o toolbox) de archivos de Matlab que permiten mandar llamar a las funciones de NI-DAQ en [14], por lo que se procedió a probar la misma, haciendo el programa Enviar.m, el cual que genera las señales que simulan al sensor CMOS pero usando las funciones de NI-DAQ en lugar de las funciones del DAQtoolbox de Matlab y se obtuvieron resultados satisfactorios.

[11] Handshaking es el intercambio de señales que permiten y coordina la comunicación entre dos dispositivos manteniendo las transferencias de datos sincronizadas

NI-DAQ cuenta con funciones que permiten hacer handshaking. Además de aprender a usar tales funciones, fue necesario entender las señales de sincronización que se utilizan para establecer dicho handshaking tanto de entrada como de salida (envío o recepción) en los dispositivos de NI. Para probar esta funcionalidad de transferencia de datos de los dispositivos de NI, se utilizó el FPGA como dispositivo de E/S (el dispositivo que interactuaría con las tarjetas de NI). Para el diseño del componente a ser implementado en el FPGA no se contaba con ninguna biblioteca de diseño para handshaking con dispositivos de NI, sino que se tuvo que hacer el código en VHDL capaz de entender estas señales de sincronización (E/S) y de transferir los datos.

Se realizó el programa Recibir.m en Matlab que utiliza (handshaking) las funciones de NI-DAQ para recibir datos, en este se utilizaron funciones que manejan transferencia de un dato a la vez.

Se estudiaron las funciones de NIDAQ que hacen handshaking transfiriendo bloques de datos (se puede especificar de que tamaño es el bloque tanto de entrada y salida) y se realizaron programas Enviarblock.m y Recibirblock.m para recibir y enviar los bloques de datos respectivamente.

Se realizaron los componentes ENVIAR02.vhd y RECIBIR01.vhd que aceptan y generan las señales requeridas para establecer el handshanking de entrada y de salida con la tarjeta de NI respectivamente, se simularon dichos componentes verificando su funcionalidad. El componente ENVIAR02.vhd envía datos desde un arreglo de memoria interno del FPGA y maneja las señales de sincronización (el protocolo de comunicación) requeridas por los dispositivos de NI

Se conectaron las tarjetas DAQPad-6507 y el FPGA FLEX10K y se verificaron las transferencias de datos entre FPGA y la tarjeta de NI, se observó que los datos recibidos por Matlab eran los enviados por el FPGA. (Se utilizaron los componentes que realizan tanto el envío como recepción de datos con handshaking).

Se midió por medio del osciloscopio la señalización handshanking para saber a qué velocidad se llevaba a cabo y se observó que la tasa de transferencia era muy lenta, en promedio de 125hz (8ms periodo), en la figura 4.2 de la sección 4.1.2 se puede observar la medición de dicha señalización.

Se pensó que esta tasa de transferencia se debía al uso de Matllab es decir que a pesar de que se manda llamar las funciones de NI-DAQ el estar dentro de la ventana de comandos de Matlab hacia lenta la respuesta por lo que se propuso utilizar otro software de aplicación para observar si habrían diferencias.

3.1.6 Evaluación de la capacidad de Visual C++ para transferencia de datos utilizando los dispositivos de NI.

Se decidió utilizar Visual C++, para realizar las mismas pruebas que con Matlab, se estudió como utilizar las funciones de NI, como agregar las bibliotecas necesarias, y se realizó el programa "DIsingleBufHandshake2.C" el cual recibe un bloque de datos. Se utilizó el componente ENVIAR02.vhd el cual que envía datos desde un arreglo fijo y que maneja las señales de handshaking para los dispositivos de NI. Se conectó la tarjeta DAQPad-6507 y el FPGA FLEX10K para verificar la transferencia de datos del FPGA a la tarjeta de NI.

Se hicieron las mediciones en el osciloscopio y se observó que los resultados fueron prácticamente los mismos (la misma tasa de transferencia) que con Matlab aun más, con Visual C++ hubo un retardo inicial en la transferencia, en la figura 4.3 de la sección 4.1.2 se puede observar la medición

Se encontró en [15] que la velocidad de transferencia del dispositivo utilizado el DAQPad-6507 es dependiente de software, "software timed" lo que quiere decir, que depende de la velocidad del procesador de la PC, memoria, velocidad del bus, del sistema operativo y que además el uso de USB lo hace mas lento que una tarjeta PCI de su misma condición.

3.1.7 Evaluación del sensor de imagen CMOS MT9T001

Se hicieron mediciones del sensor de imagen MT9T001 de todas las señales que arroja con puntas digitales de prueba para el osciloscopio de señal mixta de 16 canales. Se midieron las señales del sensor de imagen CMOS. Se observaron resultados que coincidieron con la hoja de datos. Las señales obtenidas coinciden con lo esperado (se tiene grabadas imágenes del osciloscopio donde se muestran las señales). En la figura 3.3 se muestra las señales de sincronización y datos del sensor MT9T001

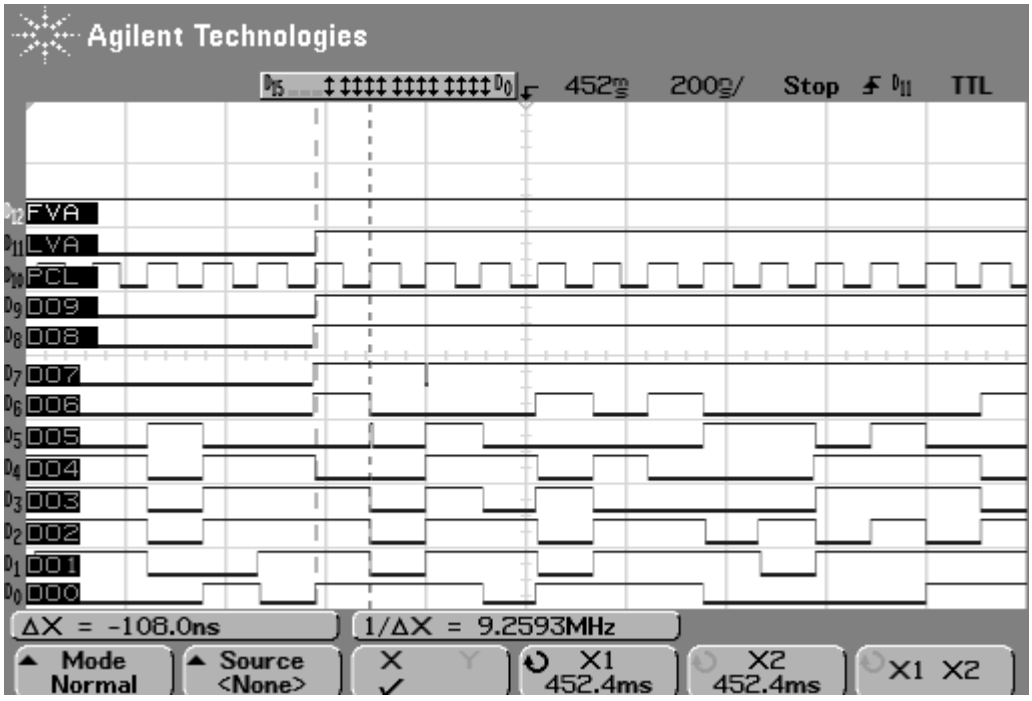


Figura 3.3 Señales de sincronización (FVA, LVA, PCL) y de las diez líneas de datos (D09-D00) del sensor MT9T001.

3.1.8 Evaluación de tarjeta DE2 con FPGA Cyclone II de ALTERA

Se obtuvo una tarjeta de desarrollo DE2 que consta de un FPGA Cyclone II de Altera, además de que cuenta con chips externos de memoria SRAM, SDRAM y FLASH, un DAC de video, interruptores, botones y display LCD. El FPGA Cyclone II EP2C35 cuenta con 105 bloques de memoria dedicados de 4K bits (483,840 bits) y 75 bloques de multiplicadores embebidos de 9bits. El Cyclone II es un ejemplo de un dispositivo FPGA moderno.

En general la memoria interna en los FPGA está limitada a pesar de que las familias nuevas de FPGA contienen bloques dedicados de memoria. Por ejemplo el FPGA FLEX10K cuenta con 9 bloques de memoria de 2K bits (18,432 bits). Como un ejemplo de la limitante de memoria de los FPGA, se explica el siguiente caso: una imagen en formato VGA consta de 640x480 píxeles, considerando píxeles de 8 bits se necesitarían 2,457,600 bits de memoria para almacenar un solo cuadro de imagen, se puede observar que los FPGA utilizados (Cyclone II, FLEX10K) no tienen suficiente memoria interna para hacer esto, por lo que se propuso aprender a utilizar la memoria SRAM (chip externo que viene en la tarjeta DE2).

La memoria SRAM 256Kx16 (4,194,304 bits) tiene un puerto de dirección y un puerto de datos bidireccional. Se estudiaron las señales requeridas para utilizar la memoria SRAM de 256Kx16 bits, se observó que se puede controlar la escritura/lectura manipulando una sola línea de habilitación de escritura (WE')

Se hizo el componente MEMFILE.vhd que se implementó en el FPGA Cyclone II, se utilizaron los interruptores, leds y botón (con filtro para rebotes) de la misma tarjeta DE2 para escribir y leer datos, ya que el puerto de datos del chip SRAM es bidireccional, el componente MEMFILE.vhd consta de un puerto bidireccional con registro de 3 estados de tal forma que para leer datos se pone alta impedancia y para escribirlos se habilita como puerto de salida (salida del FPGA entrada a la SRAM). Se observó que los datos leídos en SRAM, eran los previamente escritos.

3.1.9 Despliegue en formato VGA

Una parte importante para el proyecto fue buscar la manera de desplegar las imágenes para la verificación. Aprovechando que la tarjeta DE2 cuenta con un convertidor digital analógico (DAC) triple de 10 bits para video, se propuso utilizarlo. El ADV7123 consiste de 3 DACs (un DAC para cada color, RGB) de video de 10 bits de alta velocidad, tiene una interfaz estándar de entrada TTL y fuente de corriente de salida analógica de alta impedancia (la cual se pone a resistores para obtener la señal de voltaje analógico requerido).

Se estudiaron las señales necesarias para utilizar el DAC de video y en general para desplegar video en un monitor estándar de computadora en formato VGA, dicho formato tiene especificaciones de tiempos para la sincronización de los cuadros de imagen, se usa una señal de sincronización horizontal (se refiere a los renglones) y una vertical (se refiere a los cuadros), además de que se muestra el intervalo de tiempo válido para el despliegue de los datos. En la figura 3.4 se muestras las señales y los tiempos de sincronización y datos del formato de video VGA

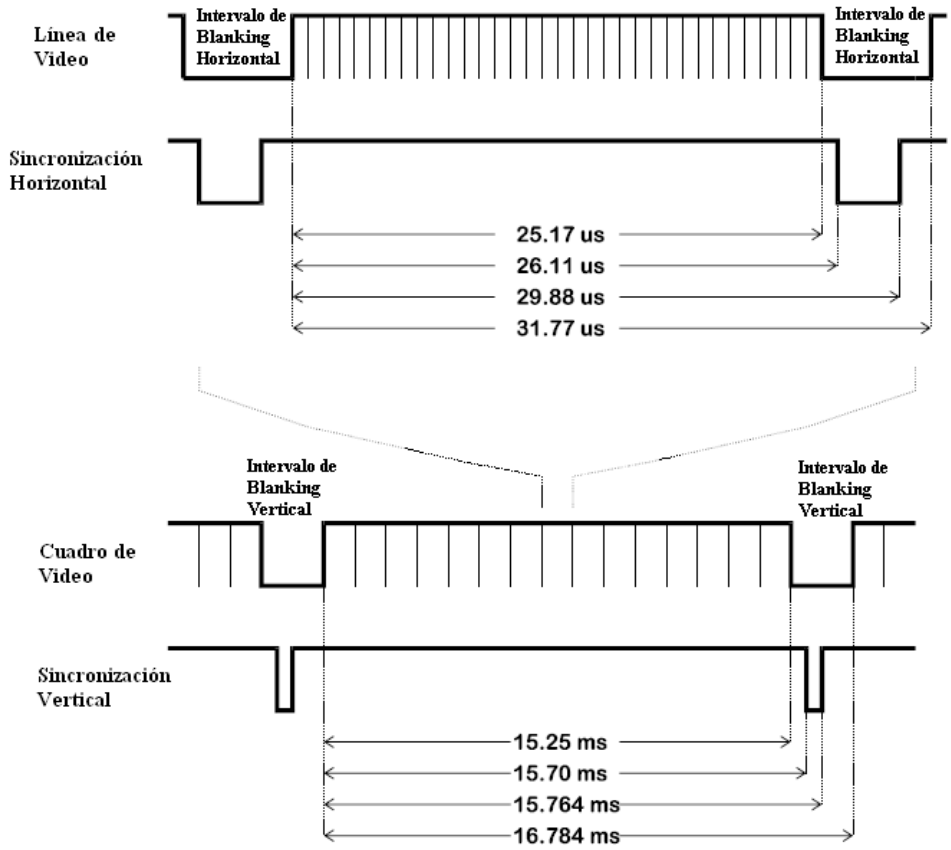


Figura 3.4 Señales y tiempos de sincronización y datos del formato de video VGA

Se realizó el componente VGA_SYNC.vhd para generar las señales de sincronización del formato VGA. Este componente está basado en el presentado en el capítulo 9 de [12], en el cual por medio de contadores se generan las señales de sincronización tomando en cuenta el orden de despliegue de los renglones y columnas del cuadro de imagen, se utilizó el reloj de la tarjeta DE2 que es de 50Mhz y se creó el componente CLKDIVIDE.vhd, el cual dividió la frecuencia a 25Mhz, ya que los tiempos del formato VGA así lo requieren.

Se verificaron las señales generadas por VGA_SYNC.vhd midiéndolas en el osciloscopio (solo estas señales).Se midieron las señales de sincronización horizontal y vertical y se observó el intervalo donde son válidos los datos RGB poniendo en "1" lógico un solo bit de los 10 del canal Rojo (del RGB)

Utilizando el mismo componente VGA_SYNC.vhd se hicieron varias pruebas donde se enviaron a las salidas RGB valores fijos, poniendo todos los bits de los 3 canales en 1's, y luego poniendo a 1's los bits de cada canal por separado y poniendo todos los bits de los canales a 0's.

Se propuso probar la capacidad de leer datos desde la memoria SRAM con vistas de poder guardar una imagen en la misma y poderla desplegar en el monitor en formato VGA

Se modificó el componente VGA_SYNC.vhd, creando el componente VGA2.vhd en el que los datos para las salidas RGB son leídos desde la memoria SRAM 256Kx16 haciendo que cada localidad de 16 bits represente 5 bits del R, 5 del G y 5 B sobrando un bit. Debido a que la SRAM tiene 18 bits de direcciones, se tienen 262,144 localidades de 16 bits; por lo tanto, en el componente mencionado se creó un contador (18 bits) del 0 al 262,143, mismo que se usó para generar las direcciones de la SRAM. Este contador se activa solo en la parte válida de la señal de video de tal forma que se pueden sacar datos solo en esta parte.

3.1.10 Almacenar datos que son enviados desde la PC en la SRAM.

Ya se contaba con el componente para escribir y leer datos del chip SRAM utilizando los switches y leds de la tarjeta DE2, sin embargo aún no se probaba escribir datos provenientes de la PC. Se propuso hacer la prueba en donde se envían datos desde la PC usando la tarjeta de NI, grabarlos en la memoria SRAM externa y verificarlos por medio de los switches y leds de la tarjeta DE2.

Siguiendo la metodología de diseño jerárquico se realizo un diseño de alto nivel llamado DATINRAM.vhd, este diseño esta formado por los siguientes componentes:

El componente DATINCTRL2.vhd acepta los datos de entrada de la PC estableciendo las señales de handshanking entre el FPGA y la tarjeta de NI, a su vez genera las señales de habilitaciones de escritura y de direcciones para ir escribiendo los datos que llegan chip de SRAM. El componente STARDATIN.vhd, espera el pulso de un pushbottun para habilitar al componente DATINCTRL2.vhd. El componente BUSES.vhd, canaliza las líneas del componente DATINCTRL2.vhd hacia el chip SRAM y una vez que ha finalizado de escribir los datos, canaliza los switches y leds de la tarjeta DE2 para poder leer los datos escritos a la SRAM. En esta prueba se envió un bloque de 16 datos desde Matlab por medio de las instrucciones usadas en el programa Enviarblock.m. En la figura 3.5 se muestra el diseño de alto nivel DATINRAM.vhd con los componentes que lo conforman

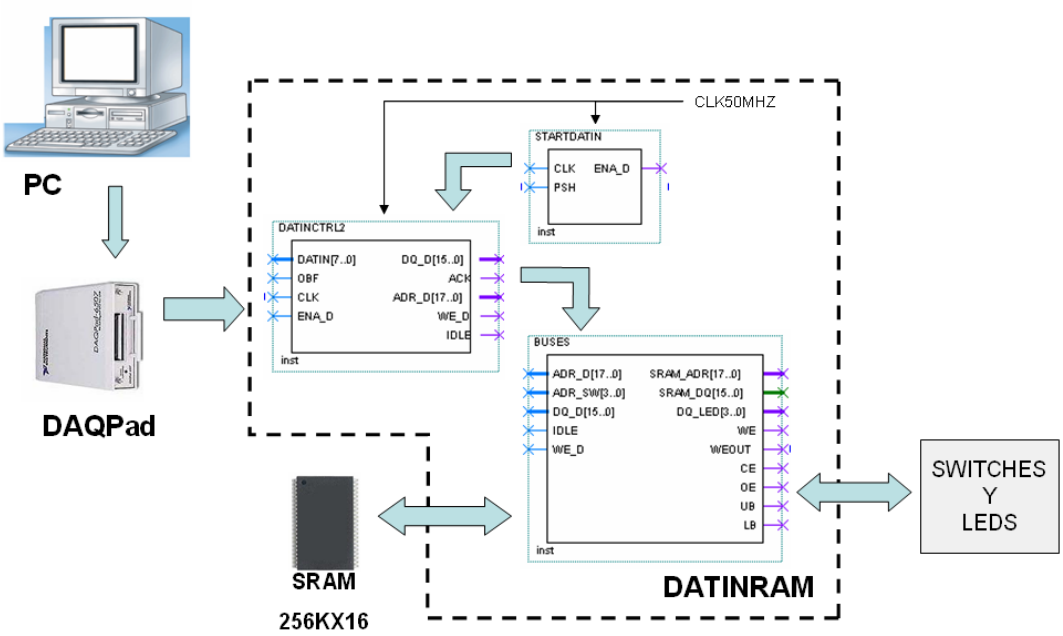


Figura 3.5 Diseño DATINRAM.vhd con los componentes que lo conforman

3.1.11 Sistema completo que manda una imagen de la PC la almacena en el chip de SRAM y la despliega en el monitor VGA

Se propuso crear un sistema que almacene los datos de una imagen enviada desde la PC al chip de memoria SRAM y una vez guardados que los despliegue en el monitor VGA.

Ya se tenían componentes en el FPGA que manejan las señales para transferencia de datos handshaking para la tarjeta de NI. Se tiene programas realizados en Matlab (y C++) usando las funciones de NI para establecer transferencia de datos handshaking. Se tiene el componente que genera las señales de sincronización y datos para el formato VGA de video, y también la versión que maneja VGA tomando los datos de entrada desde la memoria SRAM. Por lo tanto se deberían unir los componentes ya existentes o hacer otras versiones a partir de estos para formar el sistema requerido, con lo cual se cerraría el trabajo realizado para crear el ambiente de adquisición, simulación y verificación de datos.

Se realizó el programa matriz-vector.m, este separa una imagen ya existente en sus tres componentes de color (Rojo, Verde y Azul) y almacena en un vector los datos del componente rojo para posteriormente enviarlos, este programa utiliza las funciones de NIDAQ desde Matlab para establecer el handshanking para utilizar la tarjeta de NI. Se envía solo el canal rojo, ya que la biblioteca de funciones no permite hacer grupos de varios puertos, solo un puerto.

A continuación se explican cada uno de los componentes realizados para formar el diseño DESP_VGA.vhd:

El componente DATINCTRL.vhd acepta los datos de entrada de la PC estableciendo las señales de handshanking entre el FPGA y la tarjeta de NI, a su vez genera las señales de habilitaciones de escritura y de direcciones para ir escribiendo los datos que llegan a la SRAM. El componente VGA3.vhd genera las señales de sincronización necesarias para el formato VGA, también lee los datos de la SRAM generando las direcciones y poniendo en modo de solo lectura.

El componente CTRL_UNIT.vhd es una máquina de estados que se usa para habilitar a uno de los componentes anteriores a la vez, habilita al componente DATINCTRL.vhd para que escriba los datos en la SRAM y espera a que termine entonces lo deshabilita y habilita al componente VG3.vhd para que lea los datos ya escritos y los use para desplegarlos en VGA. El componente BUSCTRL.vhd canaliza las señales del componente DATINCTRL.vhd o VGA3.vhd según cual esté habilitado (controla los buses). También se utilizó el componente CLKDIVIDE.vhd, que divide la frecuencia del reloj. Cada componente se verificó por medio de las simulaciones del software CAD, QUARTUS II. Posteriormente se unieron cada uno de los componentes en un diseño de alto nivel llamado DESP_VGA.vhd.

Se probó este diseño, se enviaron por medio del DAQPad-6507 260,00 datos de la imagen mencionada que se grabaron en la SRAM y después se desplegaron en pantalla. Se logró observar la silueta de la imagen enviada, ya que era un solo componente de color. En la figura 3.6 se muestra un diagrama a bloques del diseño DESP_VGA.vhd, los componentes que los conforman y los dispositivos que interactúan con él.

Una vez que se logró formar el ambiente de trabajo, una vez que se practicó la metodología de diseño lógico programable y se logró integrar las herramientas hardware y software que se mencionaron, se propuso acoplar el sensor CMOS con el FPGA para completar la última fase del proyecto de tesis.

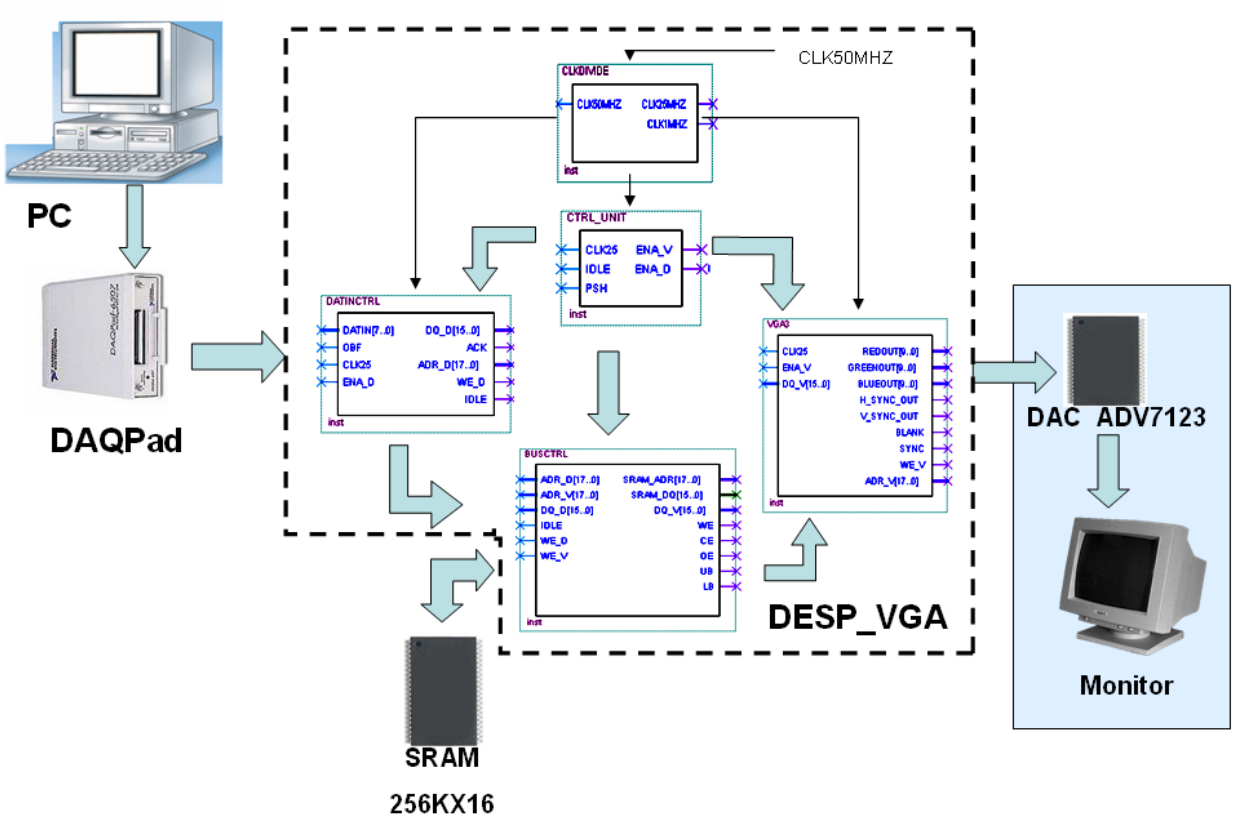


Figura 3.6 Diagrama a bloques que muestra el diseño DESP_VGA.vhd, los componentes que lo conforman y los dispositivos que interactúan

3.2 Diseño del sistema de imagen en FPGA

3.2.1 Control de los registros del sensor de imagen CMOS MT9T001 utilizando la interfase serial de dos hilos.

Como ya se mencionó en el capítulo dos, el sensor MT9T001 permite ser programado por el usuario para el tamaño de cuadro, la ubicación del cuadro y ajustes de ganancia, entre otros parámetros. Se programa a través de una interfase serial de dos hilos que controla las lecturas y escrituras a los registros internos del sensor. La comunicación serial utilizada por dicha interfase se lleva a cabo por medio del protocolo I2C, el cual es ampliamente usado para comunicación entre circuitos integrados. El protocolo I2C es de tipo maestro-esclavo. El sensor se comporta como un dispositivo tipo esclavo por lo que se propuso utilizar el FPGA como dispositivo maestro creando un diseño (un componente o varios) que establezca dicho protocolo. A continuación se explica brevemente el protocolo

Una secuencia de escritura o lectura empieza cuando el dispositivo maestro envía un bit de inicio. Después del bit de inicio, el maestro envía la dirección de 8 bits del dispositivo esclavo (sensor). El último bit de esta dirección determina si se requiere una escritura o lectura, donde “0” indica escritura y “1” indica lectura. El dispositivo esclavo reconoce su dirección enviando un bit de reconocimiento al maestro. Si se requiere una escritura, el maestro transfiere la dirección de 8 bits del registro a donde se escribirá. El esclavo manda un bit de reconocimiento para indicar que la dirección del registro ha sido recibida. El maestro transfiere un dato de 8 bits a la vez, con el esclavo enviando un bit de reconocimiento después de cada 8 bits. El MT9T001 usa datos de 16 bits para sus registros internos, por lo que requiere dos transferencias de 8 bits para escribir a un registro. Después de que 16 bits son transferidos, la dirección del registro se incrementa automáticamente de tal forma que los siguientes 16 bits sean escritos a la siguiente dirección de registro. El maestro detiene la escritura mandando un bit de inicio o un bit de paro. En la figura 3.7 a) se muestra la interfase serial de dos hilos donde se muestran las conexiones del dispositivo FPGA y el sensor MT9T001, además en la figura 3.7 b) se muestra un ejemplo de una secuencia de escritura, mostrando las señales de reloj (SCLK) y de datos (SDATA).

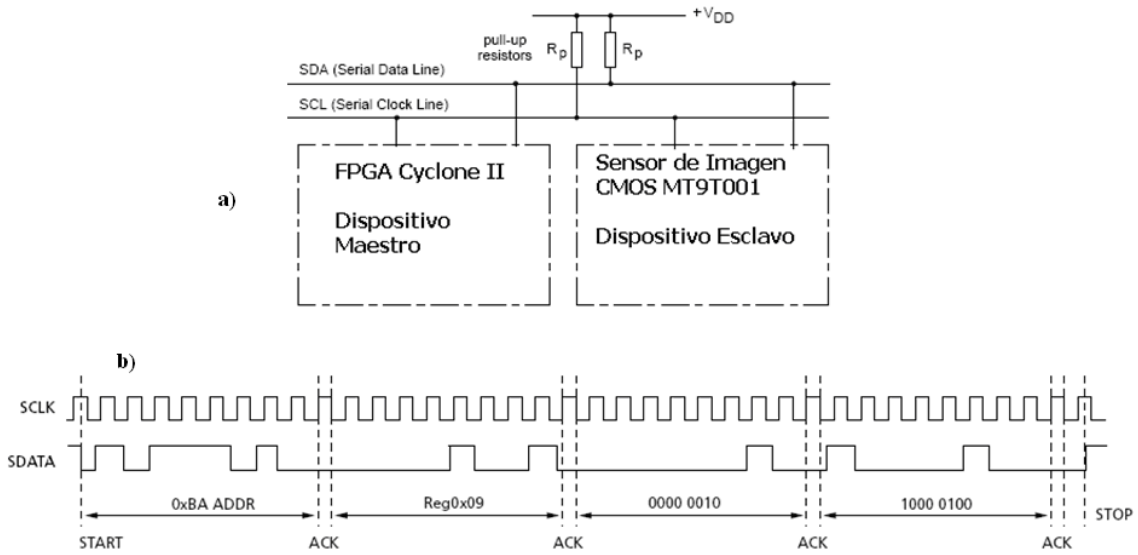


Figura 3.7 a) Interfase serial de dos hilos mostrando el FPGA y el sensor MT9T001, b) Secuencia de escritura al Reg09 con el valor 0x0284.

Ya que se pretendía modificar las funciones del sensor de imagen lo más importante era hacer escrituras a los registros internos del sensor. Por lo que se trabajó en hacer componentes para establecer el protocolo de comunicación enfocado a la secuencia de escritura.

Ya que para el caso particular del sensor MT9T001 se tiene una dirección única de dispositivo esclavo y tomando en cuenta que se quería era hacer solamente escritura, este parámetro se puede manejar como constante (que la dirección del esclavo en modo de escritura sea constante.).

Se hizo el componente I2C_WR2.vhd[12], que realiza la secuencia de escritura de 16 bits, se simuló para que hiciera un solo ciclo de escritura, los valores para la dirección del esclavo en modo de escritura y la dirección del registro son constantes dentro del código, mientras que el dato de 16 bits se dividió en dos vectores constantes de 8 bits cada uno.

Se hizo el componente I2C_WRA.vhd que es muy similar al I2C_WR2.vhd pero en este se toman los valores de la dirección del registro y de los datos a ser escritos desde afuera. Esto se hizo con el propósito de crear otro componente que le brinde dichos valores.

[12] En el apéndice B se muestran los códigos VHDL realizados de la sección 3.2.1 a la 3.2.6

Se hizo un componente I2C_CNTR.vhd que brinda al componente I2C_WRA.vhd los valores del registro y datos a ser leídos, de tal forma que en este componente se manejan como parámetros esos valores.

Se creó el diseño de alto nivel I2C_ESCRITURA.vhd que une los componentes I2C_WRA y I2C_CNTR.vhd, se simuló su comportamiento trabajando en conjunto

Para tener mayor certeza al momento de hacer la prueba en práctica, se creó otro componente I2C_SLAVE.vhd que simula la respuesta que el sensor brindaría al establecerse la secuencia de escritura.

Cabe mencionar que la línea de datos serial SDATA en el diseño I2C_ESCRITURA.vhd se declaró como un línea bidireccional y que ambas líneas SCLK y SDATA utilizan registros de 3 estados ya que la interfaz especifica que se pongan a resistencias pull up conectadas a 3.3v, por lo cual dichas líneas brindan valores de alta impedancia y de nivel de voltaje 0 (tierra).

Se integró el componente I2C_SLAVE.vhd al diseño I2C_ESCRITURA.vhd. Se simuló el comportamiento del diseño completo. Se propuso probar en la práctica dicho diseño, también se utilizó el componente para dividir la frecuencia dejando la frecuencia del reloj serial SCLK a 100khz (para cubrir los requerimientos del sensor). Se hicieron mediciones en el osciloscopio y la respuesta coincidió con la simulación.

Ya que se tenía la certeza del correcto funcionamiento del diseño, se utilizó la versión de I2C_ESCRITURA.vhd deshabilitando al componente que simulaba la respuesta que el sensor daría (I2C_SLAVE.vhd) y se propuso conectar el sensor para observar su comportamiento.

Se probó el diseño I2C_ESCRITURA.vhd conectándolo al sensor MT9T001, enviando los valores para escribir a 4 registros para cambiar el tamaño del cuadro (640x480), y para colocar la ventana (500,672). Se midieron las señales del protocolo I2C y las de sincronización del sensor donde se pudo observar que los tiempos cambiaron según los valores escritos a los registros. En la figura 3.8 se muestra el diagrama de bloques del diseño I2C_ESCRITURA.vhd.

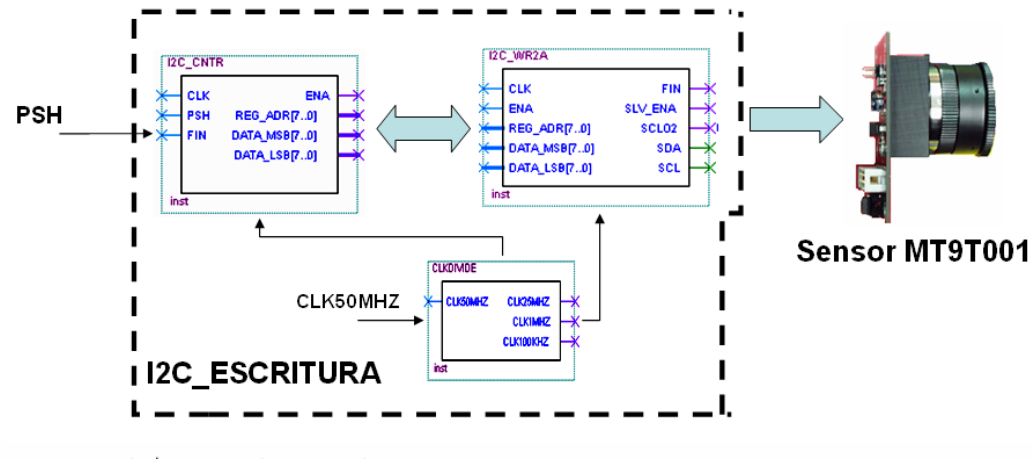


Figura 3.8 Diagrama a bloques del diseño I2C_ESCRITURA.vhd mostrando los componentes que lo conforman

3.2.2 Despliegue de un cuadro capturado del sensor (como “still” fotografía a la tasa de píxel).

Se deseaba probar la capacidad desarrollada anteriormente, por lo que se propuso desplegar en el monitor VGA un cuadro de imagen capturado por el sensor (tipo fotografía “still”), ya se había logrado desplegar un cuadro de imagen proveniente de la PC, por lo que ya contaba con un diseño en el cual basarse para lograr esto. Además debido a que se logró establecer el protocolo para escribir correctamente en los registros internos del sensor ya se tenía el control de las diferentes funciones del sensor, entre ellas las más importantes manejar el tamaño del cuadro capturado y la colocación del mismo dentro del arreglo de píxeles del sensor.

Se pretendía guardar un solo cuadro de imagen capturado en la memoria SRAM de la tarjeta DE2 y desplegar el contenido de la misma en el monitor VGA de la computadora por lo cual se mandaron comandos al sensor de tal forma que el tamaño de cuadro cupiera en la memoria SRAM. La memoria SRAM tiene 262,144 localidades de 16 bits, ya que un cuadro en formato VGA (640x480) necesita de 307,200 localidades de memoria se propuso guardar un cuadro de 640x409 (261,760 píxeles).

Se observó que para formar el diseño para desplegar un cuadro de imagen capturado por el sensor se podían reutilizar los componentes del diseño DESP_VGA.vhd hecho para desplegar un cuadro de imagen proveniente de la PC. Se debía cambiar el componente DATINCTRL.vhd que aceptaba los datos de la tarjeta de NI por otro componente que aceptara las señales de sincronización y datos del sensor MT9T001.

Se creó el componente DINCMOS.vhd, que acepta la sincronización de los datos del sensor, consta de un proceso que le permite detectar el comienzo del cuadro y otros procesos que generan las direcciones y señal de habilitación de escritura para la memoria SRAM.

Se creó el diseño de alto nivel DESP_VGA2.vhd que utiliza el componente DINCMOS y reutiliza los componentes VGA3.vhd, BUSCTRL.vhd, CTRL_UNIT.vhd y CLKDIVIDE.vhd del diseño DESP_VGA.vhd de los que ya se explicó su funcionamiento en la sección 3.1.11.

Se utilizó la tarjeta DEMO2 que es un sistema del fabricante Micron que se conecta a la tarjeta MI3100 (la tarjeta que tiene el sensor MT9T001), para evaluar los sensores de Imagen. Dicho sistema incluye el software DevWare que permite visualizar imagen del sensor y modificar sus parámetros. Se utilizó este sistema para poder visualizar la imagen que captura el sensor y así ajustar la iluminación (apertura) y el enfoque del sistema óptico de la tarjeta MI3100. En la figura 3.9 a) se muestra la tarjeta DEMO2 y la tarjeta MI3100, en la figura 3.9 b) se muestra la pantalla del software DevWare, donde se visualiza imagen del sensor y se observan los valores de los registros internos del sensor

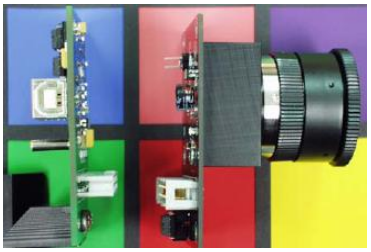


Figura 3.9 a) Tarjeta DEMO2 (izquierda) y la tarjeta MI3100 (derecha)

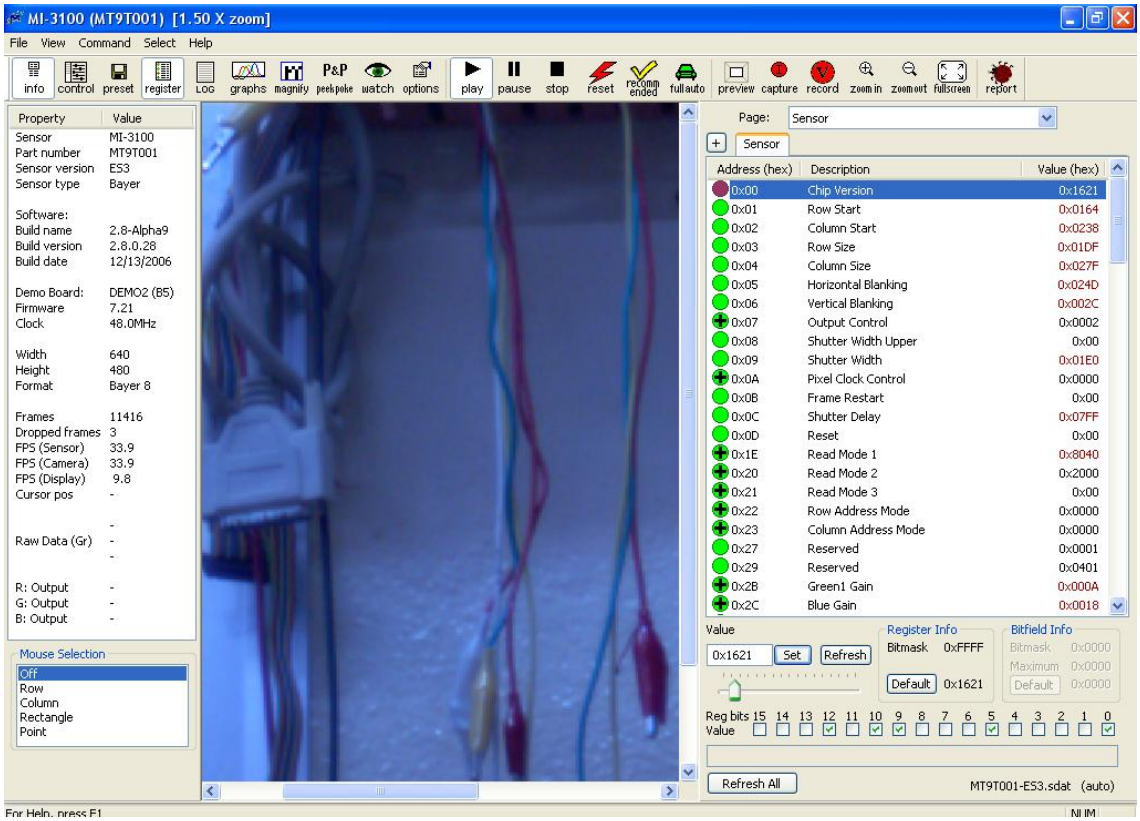


Figura 3.9 b) Software DevWare, donde se visualiza imagen del sensor (centro), y se observan los valores de los registros internos del sensor (derecha)

Una vez teniendo la certeza de que el sensor brindaría imagen con el correcto enfoque e iluminación se procedió a utilizar el diseño DESP_VGA2.vhd. Se midieron las señales de sincronización del sensor para asegurar que haya almacenado el cuadro de imagen y se observó en el monitor VGA. Para esta prueba se utilizaron los 10 bits de datos crudos del sensor y se enviaron al canal verde del DAC de video, por lo cual se pudo observar una imagen en tonos de verde en el monitor VGA (para los canales rojo y azul se pusieron valores de todos sus bits en ceros). En la figura 3.10 se muestra el diagrama de bloques del diseño DESP_VGA2.vhd y los componentes que los conforman.

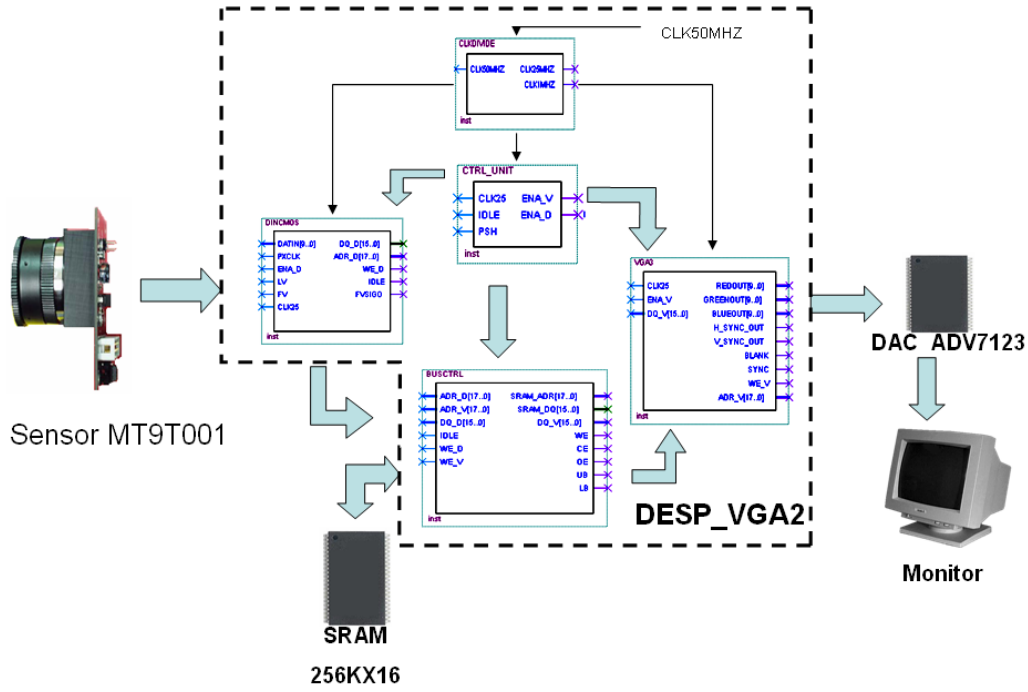


Figura 3.10 Diagrama de bloques del diseño DESP_VGA2.vhd y los componentes que los conforman

3.2.3 Diseño para realizar el tratamiento para obtener los tres componentes de color de cada píxel

Como se mencionó en la sección 2.1.2, el sensor MT9T001 usa un arreglo de filtro de color con el patrón de Bayer donde cada píxel tiene un filtro de uno de los colores primarios, por lo cual se necesita hacer un tratamiento sobre la matriz de píxeles originales para obtener los dos componentes de color faltantes de cada píxel a partir de los píxeles adyacentes. Se decidió utilizar el algoritmo de interpolación bilineal para obtener los tres componentes de color de cada píxel, debido a que es un algoritmo que requiere operaciones de promediado sobre una vecindad de píxeles, lo que permite implementarlo usando una estructura regular y repetitiva. La interpolación bilineal es un algoritmo ampliamente utilizado por su bajo costo computacional y porque brinda una calidad aceptable [8].

A continuación se explica brevemente la interpolación bilineal, si se tiene un sensor de imagen con el patrón de color de bayer como se muestra en la figura 3.11 b), se obtienen los valores de color faltantes de cada píxel haciendo el promedio de los píxeles vecinos, por ejemplo, si se esta posicionado en un píxel azul se hace el promedio de los cuatro píxeles rojos y los cuatro verdes para obtener el valor de color rojo y verde respectivamente, en la figura 3.11 a) se muestra esta idea.

Siguiendo la figura 3.11 b) se muestran a continuación algunos ejemplos de las ecuaciones requeridas:

Si se ésta en el píxel azul B_8 se obtienen los valores rojo y verde a partir de:

$$R_8 = (R_2 + R_4 + R_{12} + R_{14})/4 \text{ y } G_8 = (G_3 + G_7 + G_9 + G_{13})/4$$

Si se ésta en el píxel rojo R_{12} se obtienen los valores azul y verde a partir de:

$$B_{12} = (B_6 + B_8 + B_{16} + B_{18})/4 \text{ y } G_{12} = (G_7 + G_{11} + G_{13} + B_{17})/4$$

Si se ésta en el píxel verde G_7 se obtienen los valores rojo y azul a partir de:

$$R_7 = (R_2 + R_{12})/2 \text{ y } B_7 = (B_6 + B_8)/2$$

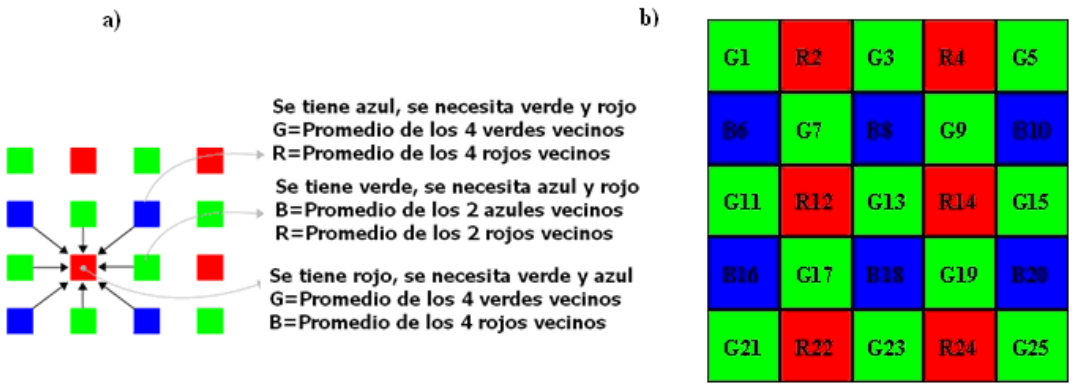


Figura 3.11. a) Explica el algoritmo de interpolación bilineal, b) muestra el patrón de Bayer

Se puede observar que para implementar estas ecuaciones en hardware se necesitan de sumadores y que las divisiones al ser en potencias de 2 se pueden implementar con corrimientos a la derecha, VHDL permite implementar las dos operaciones aritméticas. También es evidente que la operación de interpolación es del tipo operación de vecindad de ocho vecinos, siendo muy parecida a los filtros de convolución con kernel de 3X3.

Lo primero que se propuso fue probar que las operaciones aritméticas se llevaran a cabo de forma correcta por lo tanto se hizo un componente llamado ARITBILINEAL.vhd que aplica las operaciones sobre un arreglo de memoria interno de 5x5. Para aplicar las ecuaciones necesarias el componente toma en cuenta la posición del dato dentro del arreglo y dependiendo de las condiciones par e impar de renglón y columna se conoce si se encuentra en un píxel verde, rojo o azul y cuales son los componentes de color que debe estimar. Se simuló este componente cambiando los valores del arreglo de memoria y se observo que hace los cálculos correctamente.

Se estudió como hacer un componente que acomodara los datos provenientes del sensor de imagen de tal manera que se pudiera crear la tubería (“pipeline”) para aplicar la operación de interpolación bilineal. Ya que la operación de interpolación bilineal es una operación de vecindad se estudiaron las arquitecturas para implementar la operación de convolución en tubería.

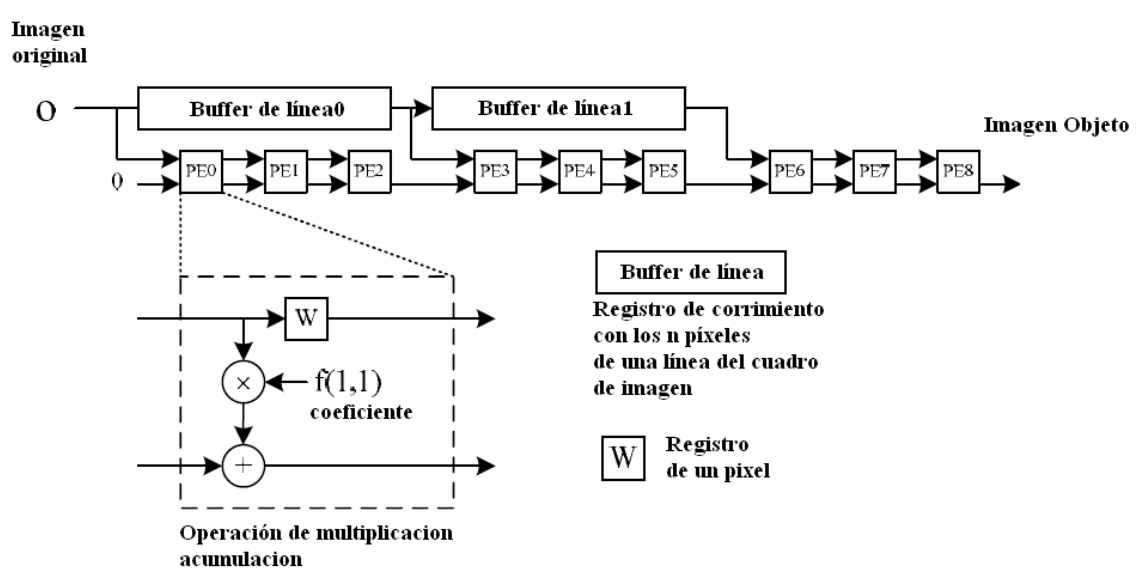


Figura 3.12 Arquitectura para implementar un convolucionador 2-D general

Se utilizó la arquitectura de un convolucionador 2-D general presentada en [16], en la figura 3.12 se muestra la misma y se puede observar que en esta sólo se requieren tener dos buffers de línea que forman la tubería, y tres registros de corrimientos de tres elementos para aplicar la ventana de convolución de 3x3 con la que se logra tener los píxeles necesarios para las operaciones de vecindad de ocho vecinos.

Se crearon los siguientes componentes para implementar la interpolación bilineal en procesamiento de tubería:

El componente PIPE.vhd crea los registros de corrimiento para formar la tubería, ya que estos registros de corrimiento deben tener un número de elementos igual al número de píxeles en una línea del cuadro de imagen y tomando en cuenta que cada píxel es de 10 bits se requiere de una capacidad de memoria considerable por lo que se utilizó un estilo de código en VHDL de tal forma que el software QUARTUS II infiera que debe implementar estos registros de corrimiento en los bloques dedicados de memoria del dispositivo FPGA sin gastar los bloques lógicos genéricos en implementar estos, en el capítulo 7 de [17] se muestran los estilos de código VHDL recomendados .

El componente CONTPOS2.vhd implementa los contadores que brindan la posición del píxel relativa al punto medio de la ventana de convolución. Esta posición se refiere a las coordenadas (renglón, columna) del píxel correspondiente al cuadro de imagen y sirve para conocer si se encuentra en un píxel con filtro verde, rojo o azul y cuales son los componentes de color que debe estimar.

El componente PIPECONV2.vhd implementa los registros para formar la ventana de convolución e implementa el hardware necesario para aplicar las operaciones aritméticas para obtener la interpolación bilineal, dependiendo de la posición del píxel este componente toma los datos requeridos desde los registros de la ventana y aplica las operaciones.

El componente FRAMEBEGIN.vhd acepta las señales de sincronización (FV, LV y PXCLK) del sensor MT9T001, para saber si se encuentra en la parte válida de los datos del sensor. Además, este componente puede ser usado por otros diseños.

Se simuló el comportamiento de cada uno de los componentes mencionados y se unieron en un componente mayor llamado PIPEBILINEAL.vhd, en la figura 3.13 se muestra el diagrama de bloques. Posteriormente se simuló el comportamiento del diseño PIPEBILINEAL.vhd, dentro de la simulación se editaron las entradas como si el sensor estuviera enviando una imagen con tamaño de 4x4 y se observó que los datos fluyeron correctamente por la tubería y los resultados de los cálculos fueron correctos.

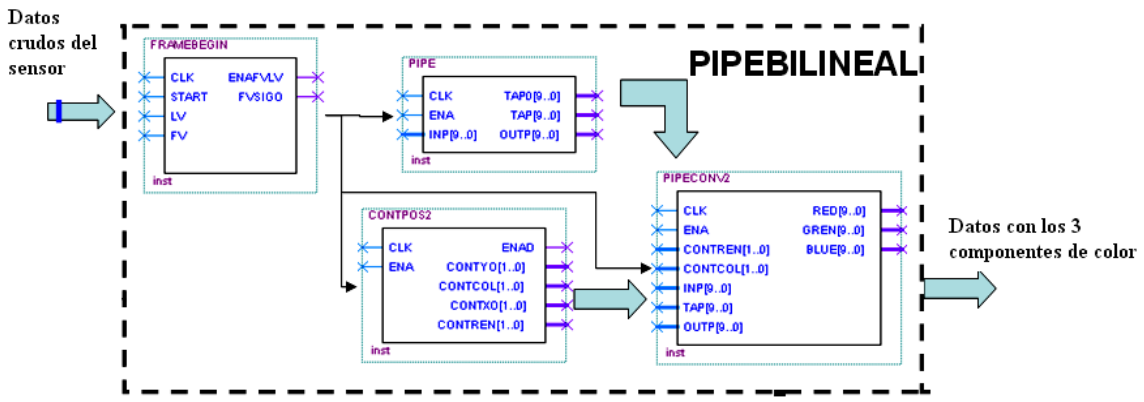


Figura 3.13 Diagrama de bloques del diseño PIPEBILINEAL.vhd, mostrando los componentes que lo conforman.

En la figura 3.14 se muestra la arquitectura implementada para realizar la interpolación bilineal en tubería, se muestra un ejemplo de una imagen de 4x4 donde se puede observar que los registros (buffers de línea) para la tubería son de 4 elementos al igual que los píxeles en las líneas del cuadro, los datos necesarios para hacer el procesamiento se almacenan en los 3 registros inferiores, estos registros forman la ventana de vecindad que se representa con la cuadrícula punteada que se encuentra sobre el cuadro de datos. El flujo de datos es de un píxel por un píxel, línea por línea. En esta arquitectura la tubería ordena los píxeles de tal forma que se pueda aplicar las operaciones en paralelo, después de un tiempo de latencia se obtienen datos procesados a la tasa de entrada del píxel.

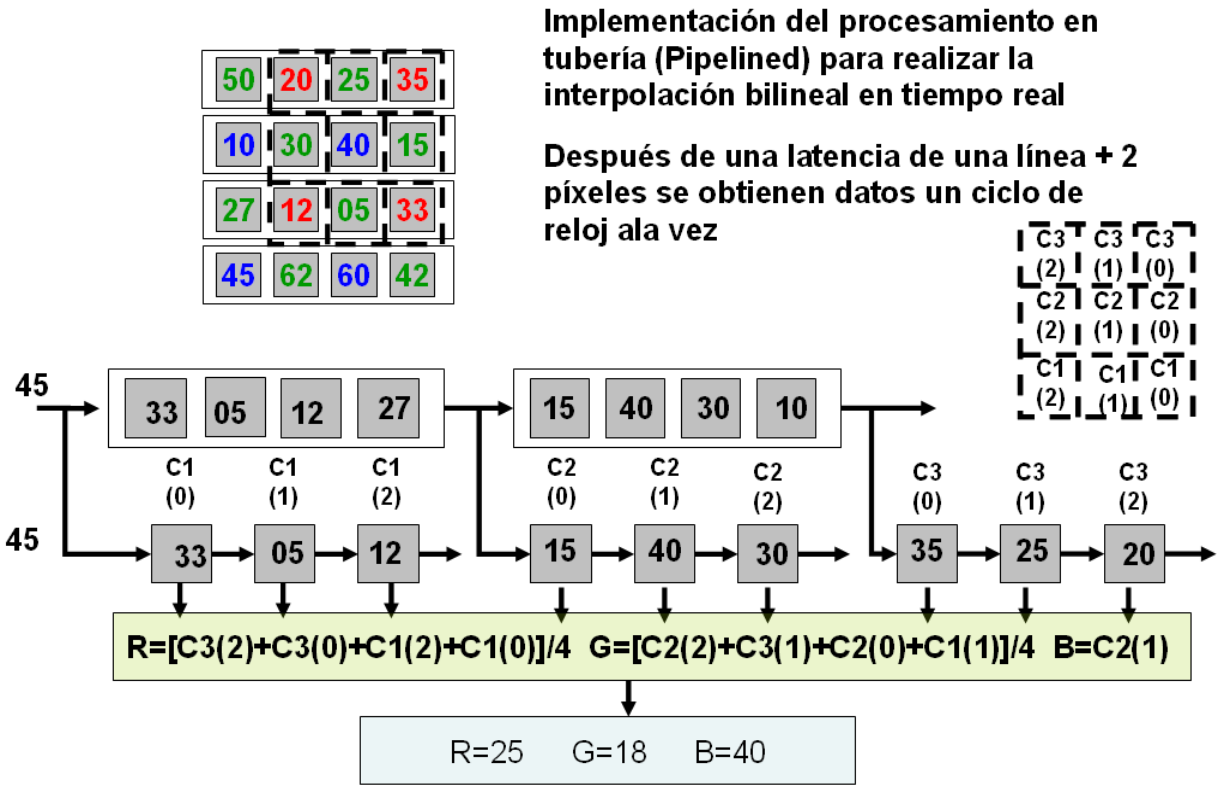


Figura 3.14 Arquitectura en tubería para implementar la interpolación bilineal, mostrándose un ejemplo donde se tiene un cuadro de imagen de 4x4.

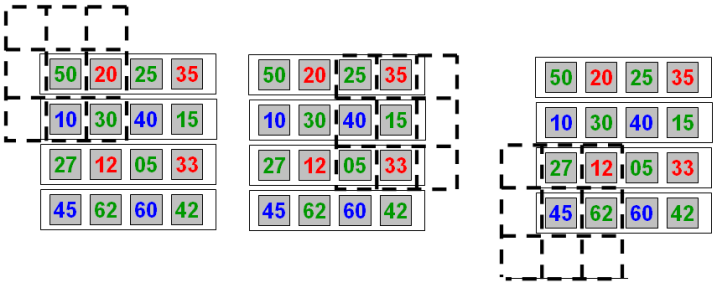


Figure 3.15 Algunos casos en los que la ventana de vecindad para aplicar el procesamiento de interpolación bilineal en tubería se encuentra en los bordes de imagen

Cabe mencionar que el componente PIPECONV.vhd, además de considerar la posición del punto medio de la ventana de vecindad para aplicar la operación pertinente, también toma en cuenta las condiciones en las que la ventana se encuentra en los bordes del cuadro de imagen, para estos casos se considera que los píxeles (datos) faltantes tienen valor cero. En la figura 3.15 se muestran algunos casos.

3.2.4 Despliegue en tiempo real de los datos crudos del sensor en un solo canal de color

Para llevar a cabo el despliegue en tiempo real de los datos (crudos) capturados por el sensor, se buscaron alternativas para utilizar el mínimo de memoria y se llegó ala conclusión de que con dos buffers de memoria, cada buffer con localidades igual al número de píxeles en una línea del cuadro de imagen sería suficiente para capturar y desplegar en tiempo real. El proceso de captura y despliegue se explica a continuación:

Se almacena la primer línea capturada en el buffer 1 (figura 3.16 a)), una vez hecho esto en el siguiente ciclo se almacena la siguiente línea capturada en el buffer 2 y simultáneamente se lee el contenido del buffer 1 para desplegar la línea (figura 3.16 b)), al siguiente ciclo se almacena la siguiente línea capturada en el buffer 1 y se despliega la línea contenida en el buffer 2 (figura 3.16c)) y así sucesivamente. En la figura 3.16, la señal ‘‘captura de línea’’ representa la señal de de línea valida del sensor y la señal ‘‘despliegue de línea’’ representa la parte válida de despliegue de línea del formato VGA

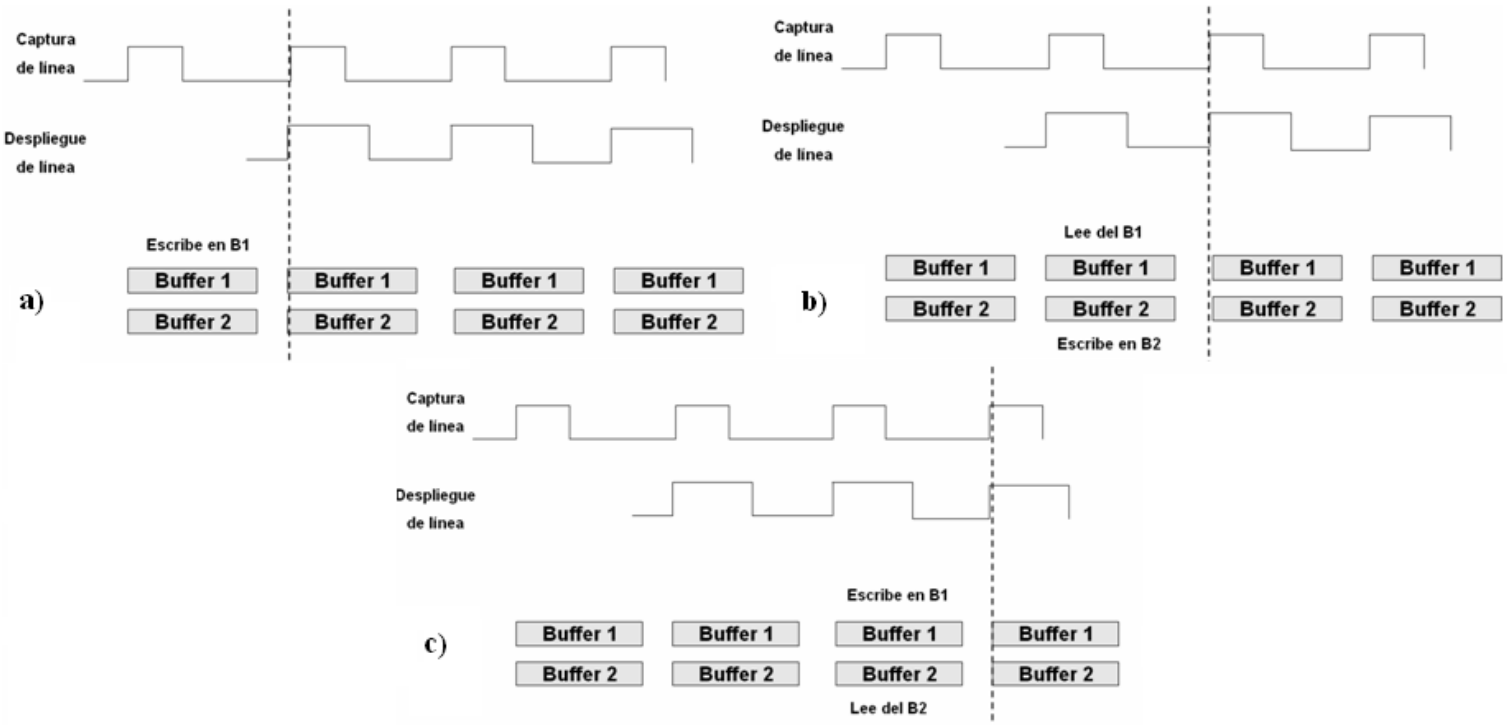


Figura 3.16 Manejo de memoria para realizar el captura y despliegue en tiempo real de los datos crudos del sensor en un solo canal de color

Cabe mencionar que para aplicar este proceso el ciclo de captura debe durar el mismo tiempo que el ciclo de despliegue es decir que lo que tarda el ciclo de captura de una línea sea lo que tarda el ciclo de despliegue de un línea y que lo que tarda en el ciclo de captura de un cuadro sea lo que tarda en el ciclo de despliegue de un cuadro. Además de tener la misma duración de tiempo estos ciclos deben estar sincronizados.

El despliegue en formato VGA (640x480) tiene especificaciones en cuanto a sus tiempos de sincronización horizontal (líneas) y vertical (cuadros), lo que da una tasa 60 cuadros/s utilizando un reloj de 25 MHZ, por otra parte el sensor MT9T001 puede usar un reloj máximo de 48 MHZ y como el tamaño de cuadro y los tiempos de blanking horizontal y vertical con lo que se pueden obtener diferentes tasas de cuadro.

Se decidió utilizar el reloj de 48 Mhz del sensor con el reloj maestro para este diseño y crear un reloj derivado de este con una frecuencia de 24Mhz que se utilizaría para el ciclo de despliegue en formato VGA; con esto se aseguró que los ciclos estuvieran sincronizados. Al utilizar el reloj de 24 Mhz se obtiene una tasa de 53 cuadros/s para el despliegue.

Se utilizó el diseño I2C_ESCRITURA.vhd para configurar el tamaño de cuadro a 640x480 y modificar los tiempos de blanking horizontal y vertical del sensor de tal forma que el ciclo de captura se empalmara con el ciclo de despliegue del formato VGA.

Se realizó el diseño DESP_REAL3.vhd que hace el despliegue en tiempo real de los datos crudos del sensor y consta de los siguientes componentes:

El componente BUFDESP3.vhd acepta las señales de sincronización del sensor, genera los contadores y señales que brindan al componente BUFLINE2.vhd las direcciones y habilitación de escritura y lectura, también habilita al componente VGA4.vhd una vez que se almacena la primera línea del cuadro de imagen.

El componente BUFLINE2.vhd crea un bloque de memoria que contiene los dos arreglos con longitud de ancho de línea del cuadro de imagen, tiene puertos, señales de habilitación y relojes separados para escritura y lectura, se usó un estilo de código VHDL para que QUARTUS II infiera memoria RAM de puerto y reloj dual y la implemente utilizando los bloques dedicados de memoria del dispositivo FPGA, como se recomienda en el capítulo 7 de [17]. En BUFLINE2.vhd se almacenan los datos capturados por el sensor y se leen par ser desplegados

El componente VGA4.vhd genera las señales de sincronización para el despliegue en formato VGA y obtiene los datos desde el componente BUFLINE2.vhd poniéndolos en el canal verde.

El componente PSH.vhd genera una señal de inicio para el diseño DESP_REAL3.vhd por medio de un pushbutton y el componente CLKDIVIDE.vhd divide la frecuencia del reloj principal 48Mhz para obtener el de 24mhz.En la figura 3.17 muestra un diagrama de bloques del diseño DESP_REAL3.vhd

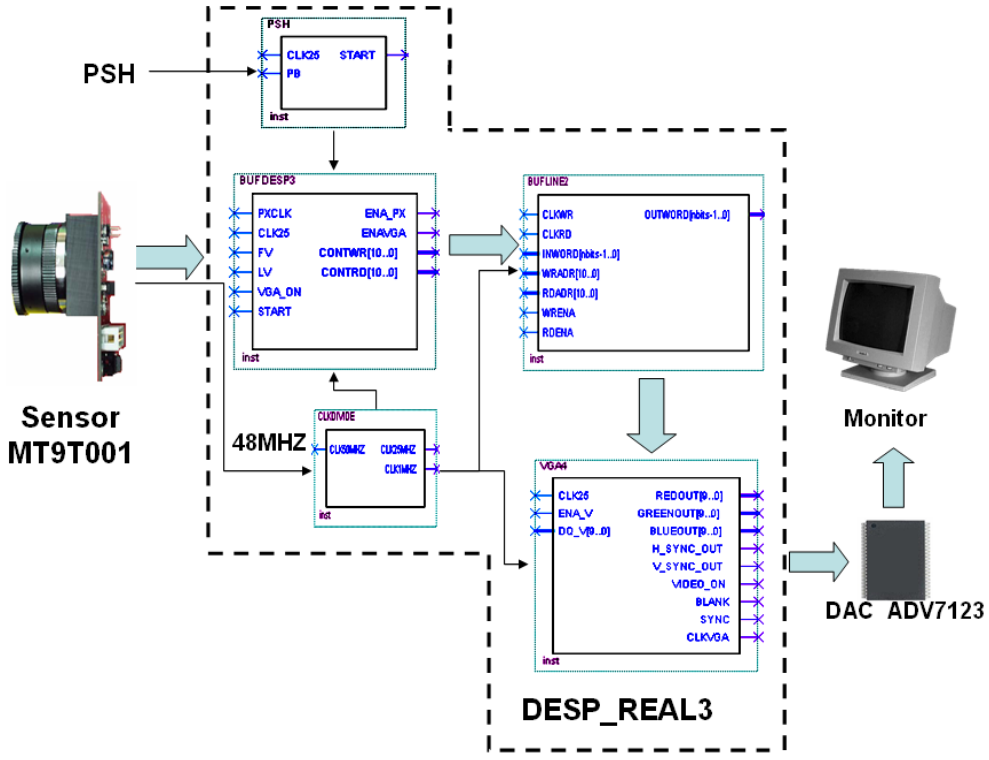


Figura 3.17 Diagrama a bloques del diseño DESP_REAL3.vhd, mostrando los componente que lo conforman

3.2.5 Despliegue en tiempo real de los datos interpolados en los 3 canales de color

Una vez logrado el despliegue en tiempo real de los datos crudos del sensor en un solo canal de color, se propuso hacer el despliegue en tiempo real en los tres canales de color, de los datos interpolados es decir, los datos obtenidos al aplicar el proceso de interpolación bilineal para obtener los componentes de color faltantes en cada píxel. El diseño PIPEBILINEAL.vhd aplica las operaciones de la interpolación bilineal en tubería (sección 3.2.3) y el diseño DESP_REAL3.vhd despliega en tiempo real los datos capturados por el sensor en un solo canal de color (sección 3.2.4), se trabajó en una versión nueva de DESP_REAL3.vhd llamada DESP_REAL4.vhd

El diseño DESP_REAL4.vhd crea tres arreglos de memoria, cada arreglo con dos buffers de memoria, cada buffer con un número de localidades igual al número de píxeles en una línea del cuadro de imagen, los tres arreglos de memoria se utilizan para hacer el proceso de captura y despliegue en tiempo real para los tres componentes de color. El proceso de captura y despliegue en DESP_REAL4.vhd se hace de manera similar que en DESP_REAL3.vhd sólo que toma que en cuenta el tiempo de latencia inicial que tarda el diseño PIPEBILINEAL.vhd en brindar los datos interpolados y se manejan los tres arreglos de dos buffers en vez de un solo arreglo. Como se observa en la figura 3.18 los datos comienzan a ser desplegados después de latencia del procesamiento de interpolación y de haber almacenado la primera línea de datos interpolados.

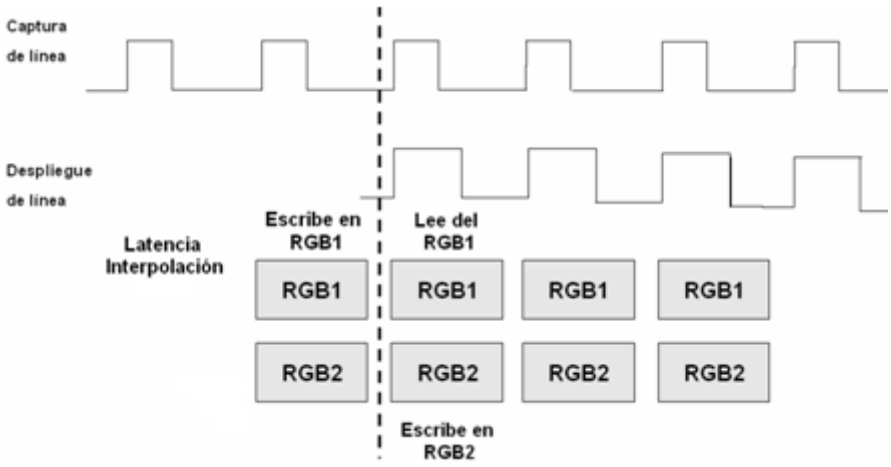


Figure 3.18 Manejo de memoria para realizar la captura y despliegue en tiempo real de los datos interpolados en un los tres canales de color. Los bloques RGB1 y RGB2 representan los buffers para almacenar los datos

Se unieron los diseños PIPEBILINEAL.vhd y DESP_REAL4.vhd para formar el diseño DESP_BAYER.vhd. A continuación se explican los componentes que los conforman:

Al diseño PIPEBILINEAL.vhd se le agregó el componente PSH.vhd para iniciar el proceso de interpolación, los componentes PIPE.vhd, PIPECONV2.vhd, FRAMEBEGIN.vhd y CONTPOS2.vhd se comportan como ya han sido descritos en la sección 3.2.3. Al componente CONTPOS2.vhd se le añadió una señal que habilita a DESP_REAL4.vhd después que se tiene listos los primeros datos procesados.

En el diseño DESP_REAL4.vhd el componente BUFDESP4.vhd es una versión modificada del componente BUFDESP3.vhd (sección 3.2.4) y realiza la misma función sólo que espera la señal de habilitación proveniente del diseño PIPEBILINEAL.vhd y controla las lecturas y escrituras a los componentes BUFRED.vhd, BUFGREEN.vhd y BUFBLUE.vhd.

Los componentes BUFRED.vhd, BUFGREEN.vhd Y BUFBLUE.vhd crean cada uno un bloque de memoria como el componente BUFLINE2.vhd (sección 3.2.4), cada componente almacena uno de los 3 canales de colores de los resultados interpolados.

EL componente VGA5.vhd realiza lo mismo que VGA4.vhd solo que acepta los 3 puertos de datos provenientes de los componentes BUFRED.vhd, BUFGREEN.vhd y BUFBLUE.vhd, de tal forma que brinda el formato de despliegue VGA en los tres canales de color. En la figura 3.19 se muestra el diagrama de bloque donde interactúan los diseños PIPEBILINEAL.vhd y DESP_REAL4.vhd para formar el diseño DESP_BAYER.vhd.

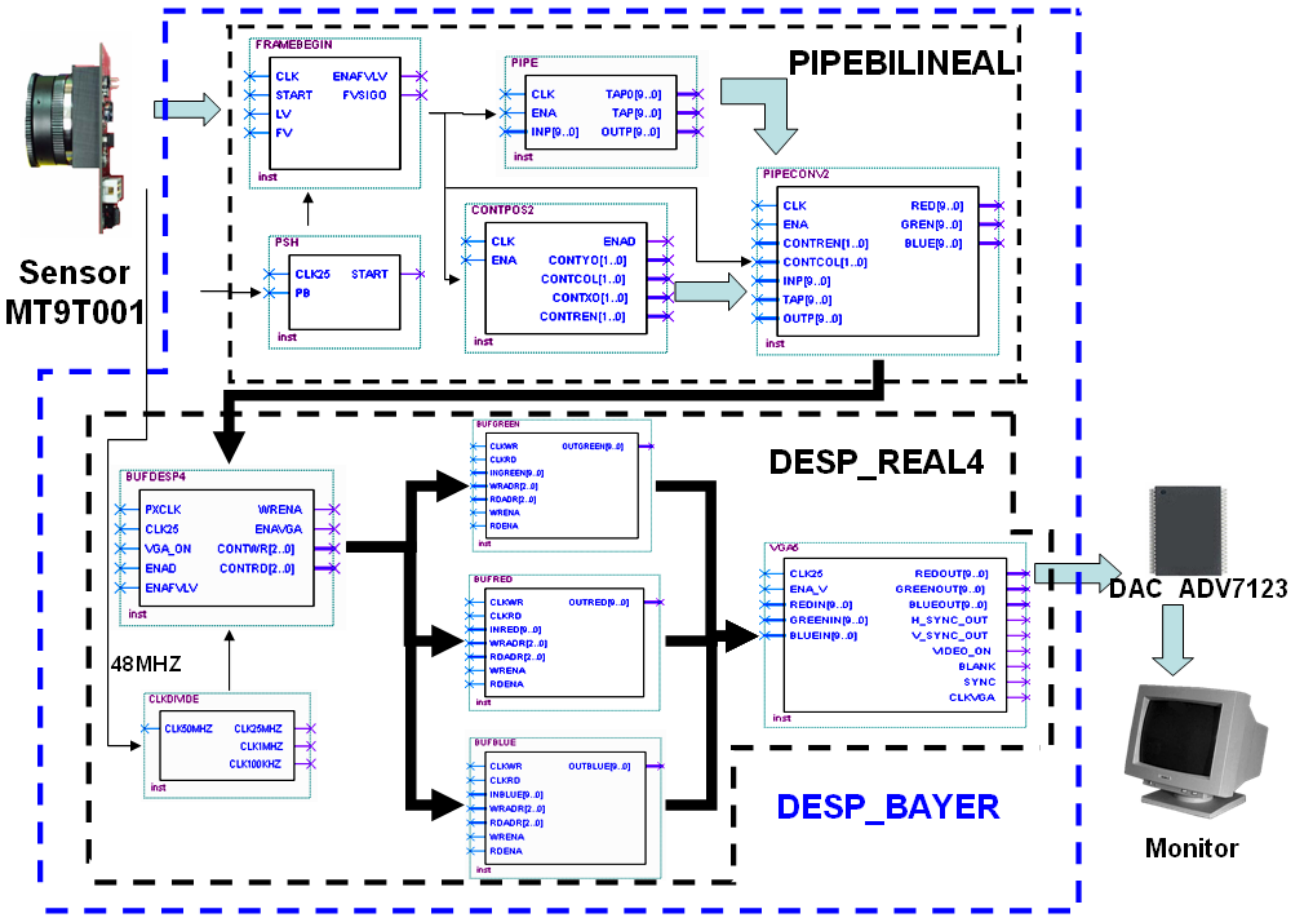


Figura 3.19 Diagrama a bloques del diseño DESP_BAYER.vhd mostrando los componentes que lo conforman

3.2.6 Corrección o balance del color

La obtención de los tres componentes de color de cada píxel por medio de la interpolación bilineal u otros métodos no asegura que la rendición o fidelidad del color en un monitor sean percibidas por el ser humano de forma adecuada. Se debe aplicar una corrección de color ya que la respuesta espectral del sensor CMOS a la luz es distinta que la respuesta del ojo humano y distinta de la respuesta del dispositivo de despliegue [18].

La corrección de color se realiza multiplicando una matriz de 3x3 con el vector formado por los valores R, G y B de cada píxel interpolado, como se muestra a continuación:

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} r_1 & r_2 & r_3 \\ g_1 & g_2 & g_3 \\ b_1 & b_2 & b_3 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}, \text{ o } \quad \begin{aligned} R' &= r_1R + r_2G + r_3B \\ G' &= g_1R + g_2G + g_3B, \text{ donde} \\ B' &= b_1R + b_2G + b_3B \end{aligned}$$

$\begin{bmatrix} R \\ G \\ B \end{bmatrix}$ = valores R, G y B del píxel interpolado, $\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}$ = valores R, G y B del píxel corregido y $\begin{bmatrix} r_1 & r_2 & r_3 \\ g_1 & g_2 & g_3 \\ b_1 & b_2 & b_3 \end{bmatrix}$ es la matriz de corrección.

Típicamente los coeficientes de la matriz de corrección tienen valores $r_1 > 1, r_2 < 1, r_3 < 1$, $g_1 < 1, g_2 > 1, g_3 < 1$, $b_1 < 1, b_2 < 1, g_3 > 1$. Donde la suma de los valores de cada juego de tres coeficientes debe ser igual a 1 para mantener el balance del color [19].

A continuación se muestra un ejemplo de los coeficientes para obtener el valor R', $r_1 = 1.6, r_2 = -.03, r_3 = -0.3$, $R' = 1.6R - 0.3G - 0.3B$. El efecto de esta ecuación es resaltar el componente rojo y substraer la porción de los componentes verde y azul.

Se buscaron varias alternativas para implementar las operaciones de multiplicación y adición/sustracción necesarias. Ya que el dispositivo FPGA Cyclone II cuenta con bloques de multiplicadores embebidos se utilizaron para implementar los productos. Los coeficientes de la matriz de corrección de color pueden ser números enteros con parte fraccionaria por lo que se buscó la manera de realizar las operaciones de productos de estos valores.

Se realizaron las operaciones para implementar la corrección de color de de la siguiente manera: Se obtiene el producto de la parte fraccionaria del coeficiente por el dato interpolado (R, G o B), el producto de la parte entera del coeficiente por el dato interpolado y se suman dichos productos resultando un número entero (despreciando la parte fraccionaria). Esto se hace para cada operación de producto de dato interpolado por coeficiente, estas operaciones se suman dando un resultado que se limita a un máximo posible y a cero si es un resultado negativo. Cabe mencionar que para realizar estas operaciones se usó la representación binaria sin signo de los coeficientes, es decir, la representación binaria de la parte fraccionaria y de la parte entera del coeficiente. En la figura 3.20 se muestran las operaciones para obtener la ecuación para el rojo corregido (R'), para obtener las ecuaciones para el verde y el rojo corregido (G',B') se usa la misma estructura utilizando los coeficientes correspondientes.

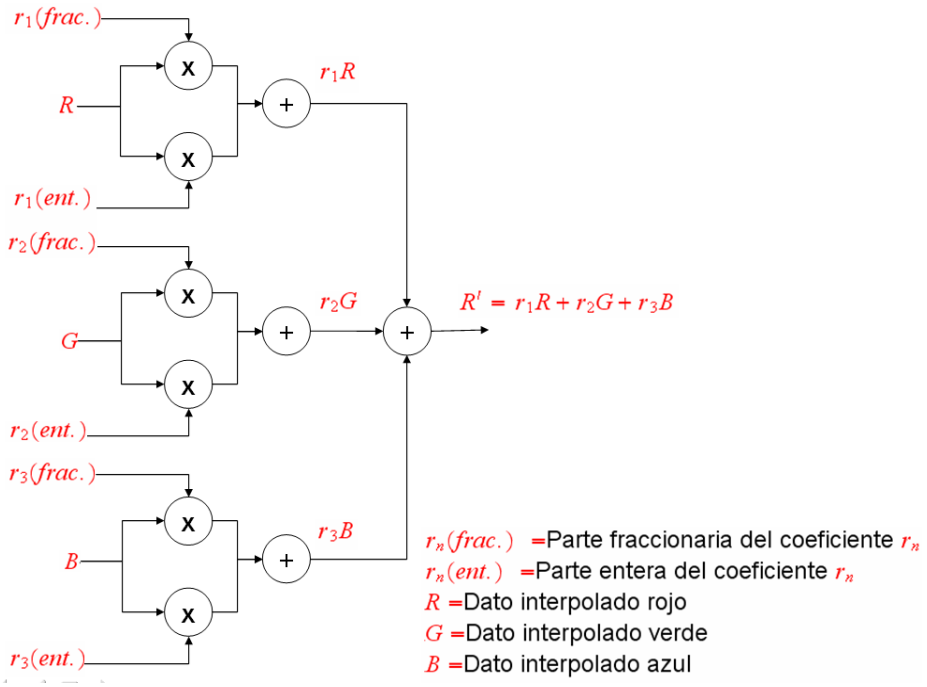


Figura 3.20 Diagrama que muestra la arquitectura para implementar las operaciones para obtener la ecuación para el componente rojo corregido R'.

Se creó el componente MULTADD5.vhd que implementa las operaciones mencionadas anteriormente, en la simulación se utilizaron distintos valores de los datos RGB y para la matriz de corrección se copiaron los coeficientes que pone el software de la tarjeta de evaluación del sensor, se observó que los cálculos obtenidos fueron correctos. Este diseño forma el esqueleto necesario para aplicar la matriz de corrección, sin embargo la búsqueda de los coeficientes depende de cada caso particular

Se creó un nuevo diseño llamado DESP_BAYER2.vhd que une el componente MULTADD5.vhd con el componente PIPEBILINEAL.vhd que realiza la interpolación en tubería y al componente DESP_REAL4.vhd que lleva a cabo el despliegue en tiempo real de los 3 canales de color. En la figura 3.21 se muestra un diagrama a bloques.

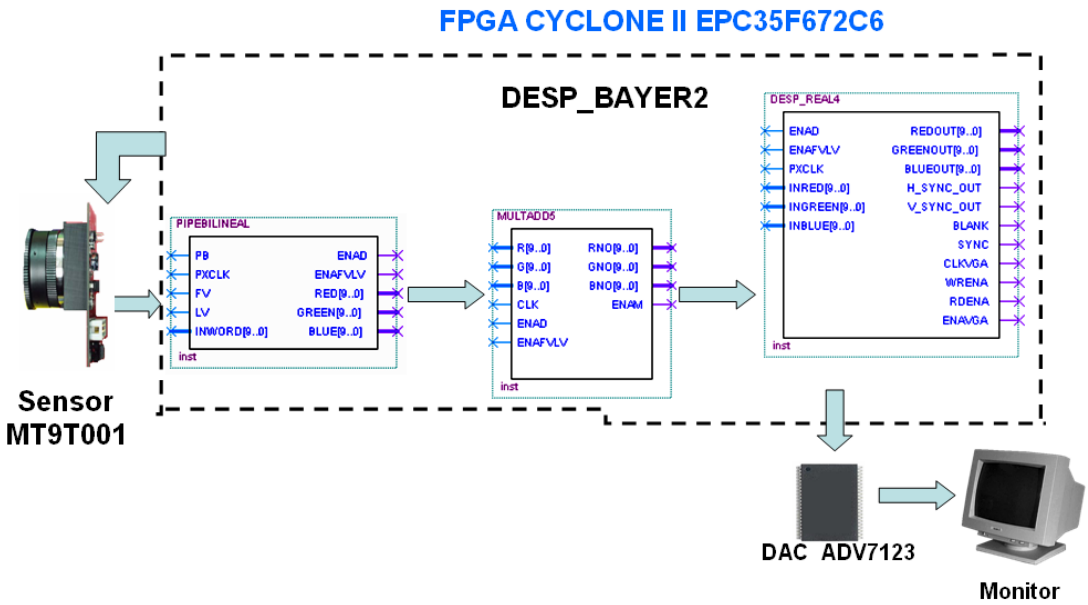


Figure 3.21 Diagrama a bloques del diseño DESP_BAYER2.vhd, mostrando los componentes que lo conforman.

Capítulo IV

Resultados

Se presentan los resultados obtenidos tanto del ambiente de desarrollo como del diseño de la cámara inteligente. Los resultados se muestran a través de diagramas de tiempos obtenidos por el osciloscopio, además de los códigos en VHDL, Matlab y Visual C++ que se muestran en los apéndices A, B y C.

4.1 Resultados sobre el ambiente de desarrollo

4.1.1 Simulación de las señales del sensor

Los siguientes resultados se refieren a la configuración para ejercitar el FPGA para que fuera capaz de aceptar las señales de sincronización y datos del sensor de imagen CMOS (ver sección 3.1.3). Se hicieron los componentes (códigos en VHDL) para aceptar las señales de sincronización y datos del sensor, resultados de la simulación correctos.

Los siguientes resultados se refieren a la generación de las señales que simulan las del sensor (ver sección 3.1.4). Se hicieron programas en Matlab para generar las señales que simulan las del sensor. Se generaron las señales que simulan las del sensor por medio de la PC utilizando Matlab y tarjeta de NI como se explicó en 3.1.4, se probaron los componentes realizados en la sección 3.1.3, se hicieron mediciones en el osciloscopio.

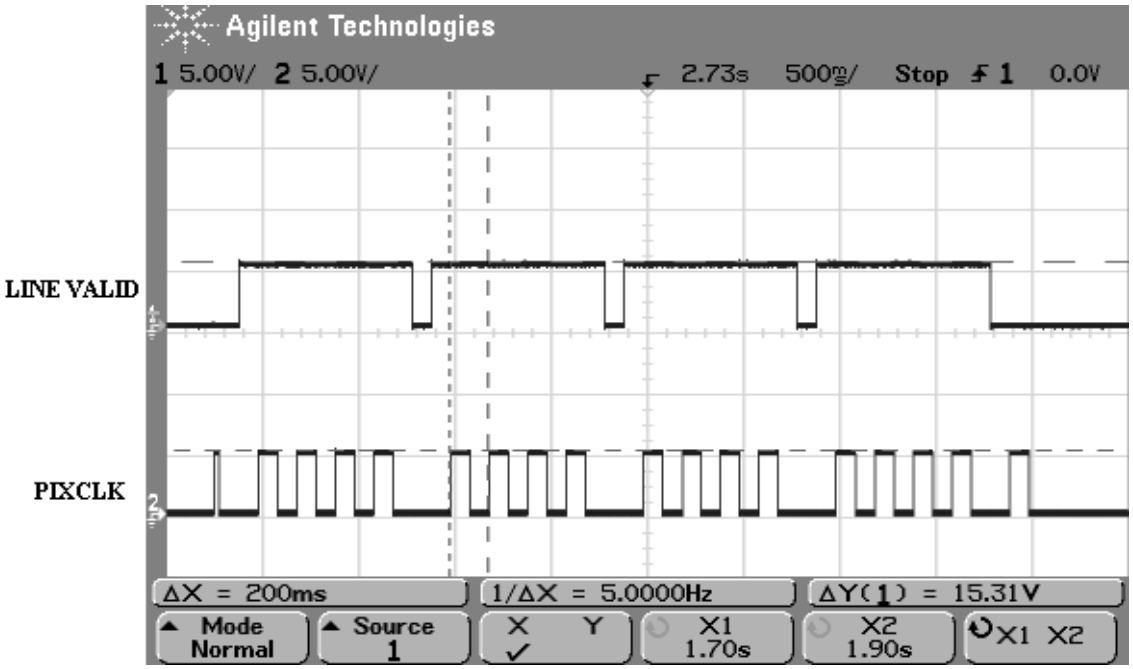


Figure 4.1 Señales generadas por la tarjeta de NI, que simulan las señales de LV y PXCLK del sensor MT9T001

4.1.2 Resultados de la evaluación de transferencia de datos entre la PC y el FPGA y la evaluación del sensor MT9T001

Para conocer la capacidad de transferencia de datos de Matlab con dispositivos de NI, se realizaron programas utilizando NI-DAQ desde Matlab para recibir y enviar los bloques de datos. Se realizaron componentes para el FPGA de tal forma que generaran y aceptaran las señales requeridas para establecer el handshanking tanto de entrada como de salida con la tarjeta de NI, ver sección 3.1.5.

Se midió por medio del osciloscopio la señalización handshanking para saber a que velocidad se llevaba a cabo y se observó que la tasa de transferencia era muy lenta, en promedio de 125hz (8ms periodo). En la figura 4.2 se muestra la señalización de handshaking (STB, IBF) de la tarjeta de NI para recibir datos, se tiene grafica del osciloscopio donde se muestran alrededor de 4 datos transferidos (DATOS), se utilizo el componente que envía datos del FPGA a la PC. Para entender el protocolo handshaking utilizado por la tarjeta de NI DAQPad-6507, se puede consultar el capítulo 3 de [20].

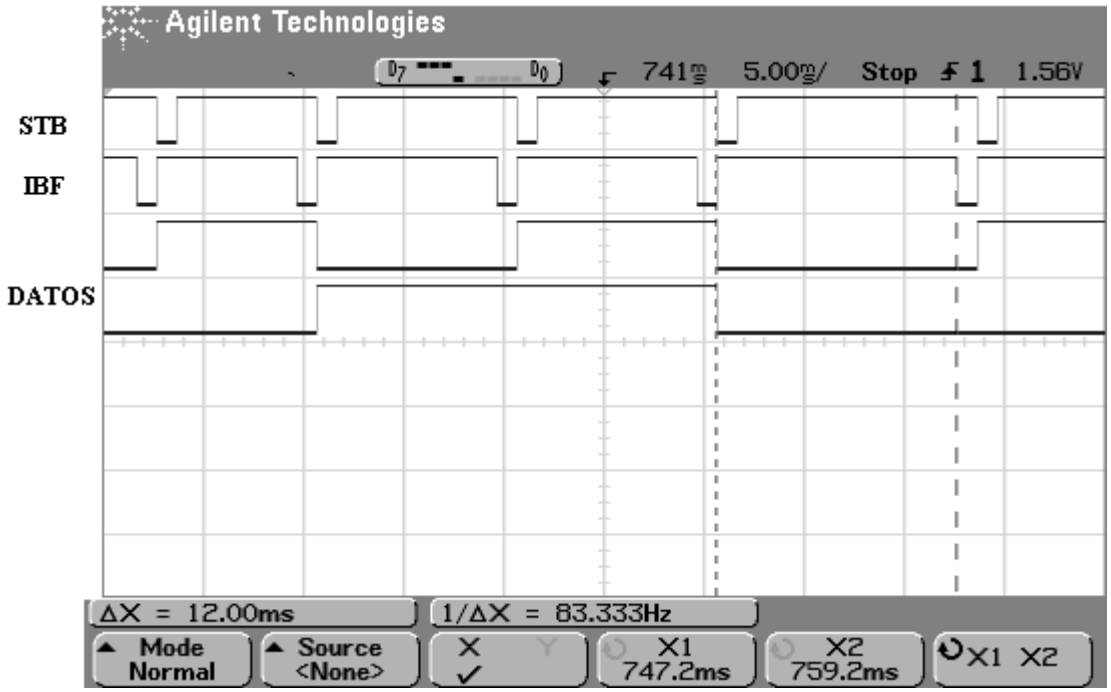


Figure 4.2 Señales de handshaking de para transferencias de entrada IBF y STB, dos líneas de datos.

Por otro lado y con relación ala sección 3.1.6, se utilizó Visual C++ para verificar la transferencia de datos entre el dispositivo de NI y el FPGA, se hicieron las mediciones en el osciloscopio y se observó que los resultados fueron prácticamente los mismos (la misma tasa de transferencia promedio de 125hz) que con Matlab aun mas, con Visual C++ hubo un retardo inicial en la transferencia. En la figura 4.3 se muestra la señalización de handshaking (STB, IBF) de la tarjeta de NI para recibir datos, se puede observar que existe un tiempo de 28.2ms en el que la transferencia no se lleva a cabo de forma correcta, después de este tiempo, se trasfieren 4 datos (DATOS) en donde las señales de handshaking (STB, IBF) se comportan correctamente, se utilizo el componente que envía datos del FPGA a la PC.

Se encontró que la velocidad de transferencia del dispositivo de NI utilizado el DAQPad-6507, es dependiente de software, "software timed", lo que quiere decir, que depende de la velocidad del procesador de la PC, memoria, velocidad del bus, del sistema operativo y que además el uso de USB lo hace mas lento que una tarjeta PCI de su misma condición.

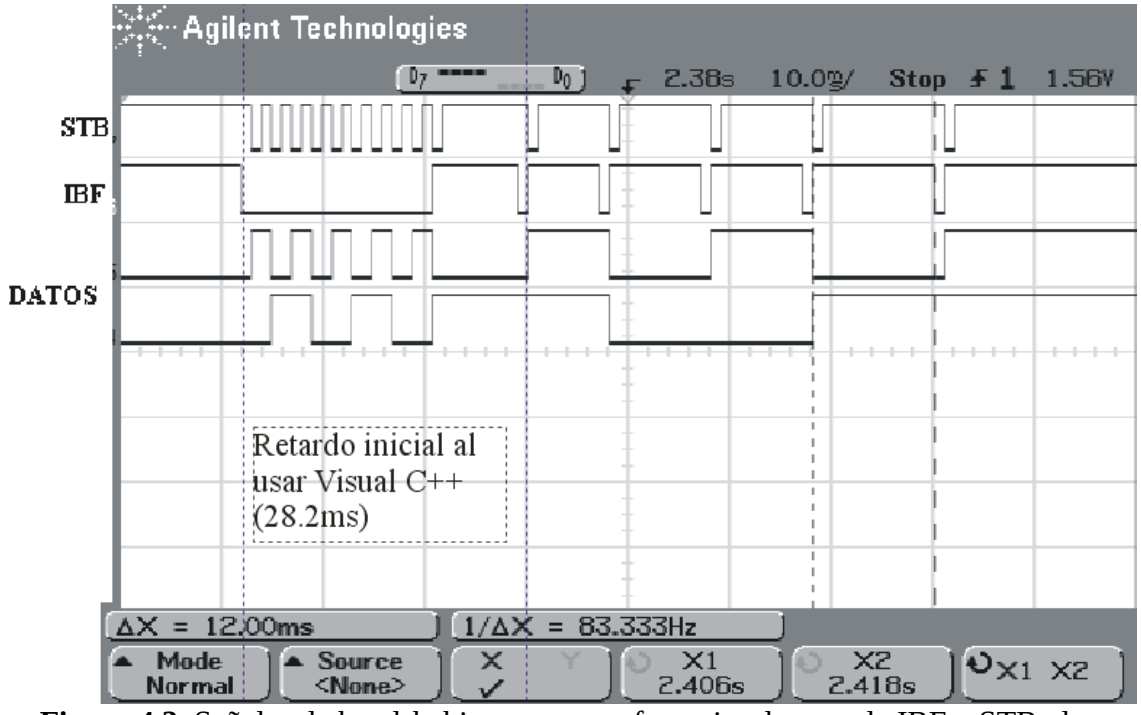


Figure 4.3 Señales de handshaking para transferencias de entrada IBF y STB, dos líneas de datos, con un retardo inicial en la transferencia

Se concluye que si se utiliza una tarjeta de entradas/salidas digitales de NI no dependiente de software, es decir una "hardware timed"[13], se pueden alcanzar tasas de transferencias convenientes y bien definidas por lo que se podrían realizar las mismas transferencias de datos expuestas en 3.1.5 y 3.1.6 (en tasas de datos de hasta 10Mhz usando un dispositivo de la serie M) [21].

Se hicieron mediciones de las señales de sincronización y datos del sensor MT9T001 y coincidieron con lo especificado en la hoja de datos. En la figura 4.4 se muestra las señales de sincronización y datos del sensor MT9T001 (Ver sección 3.1.7)

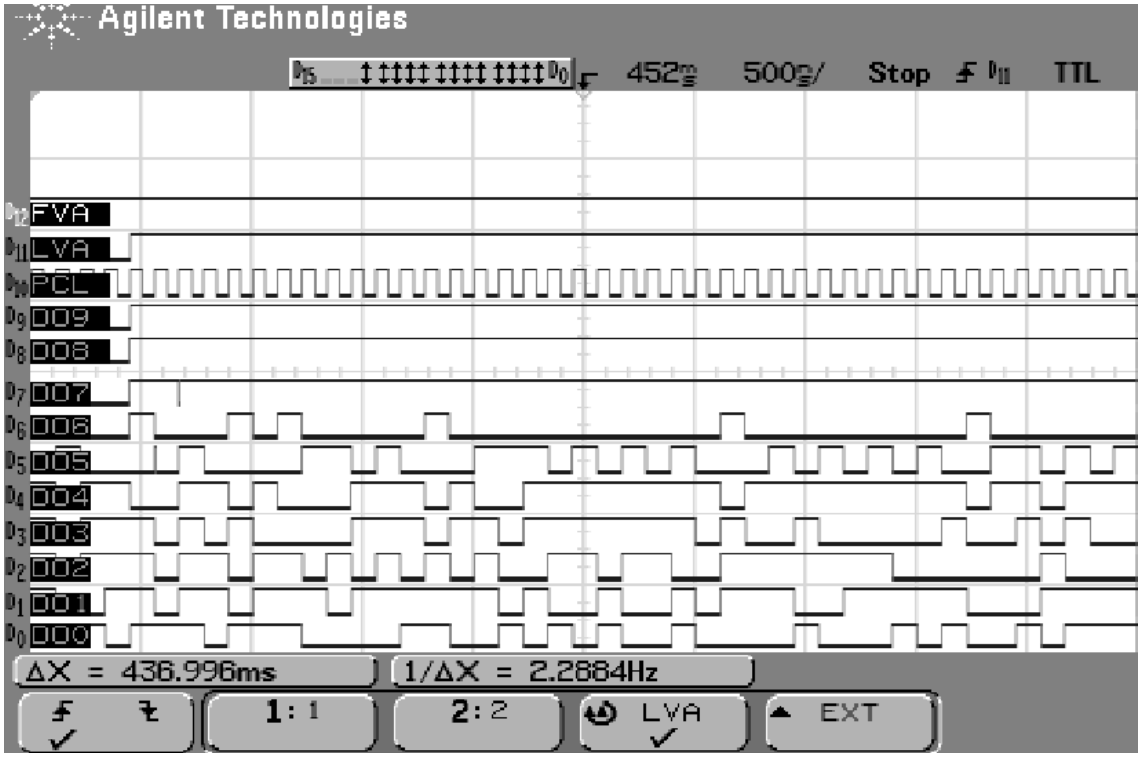


Figura 4.4 Medición de señales de sincronización (FVA, LVA, PCL) y de las diez líneas de datos (D09-D00) del sensor MT9T001.

[13] Un dispositivo "hardware timed" genera señales basado en un reloj (una base de tiempos). Un dispositivo "software timed" no tiene reloj, lo que limita la actualización de su tasa y no es deterministico

4.1.3 Despliegue en formato VGA de datos en memoria SRAM

Se evaluó la capacidad de lectura/escritura del chip SRAM del tablero de desarrollo DE2 del FPGA (ver sección 3.1.8). Se aprendió a utilizar el chip externo de memoria SRAM, se hizo componente para utilizar el chip SRAM por medio de los botones y leds de la tarjeta DE2 y observó que los datos leídos en SRAM, eran los previamente escritos.

Para el despliegue en formato VGA, se aprendió a utilizar el chip DAC para VGA de la tarjeta DE2 y en general las señales para desplegar video en monitor estándar de computadora en formato VGA. Se realizó un componente para generar las señales de sincronización para formato VGA y se midieron. En la figura 4.5 se muestra las señales de sincronización vertical y horizontal generadas para el formato VGA.

Utilizando el mismo componente se hicieron varias pruebas donde se enviaron a las salidas RGB valores fijos. Se observaron en la pantalla el color verde, rojo, azul, blanco y negro que corresponden a los valores dados en cada prueba. Se modificó el componente que genera las señales para VGA, en este los datos para las salidas RGB son leídos desde la memoria SRAM 256Kx16 haciendo que cada localidad de 16 bits represente 5 bits del R, 5 del G y 5 B sobrando un bit. La prueba fue exitosa, se logró observar un patrón de colores (unas franjas de colores) en el monitor VGA. (Ver sección 3.1.9).

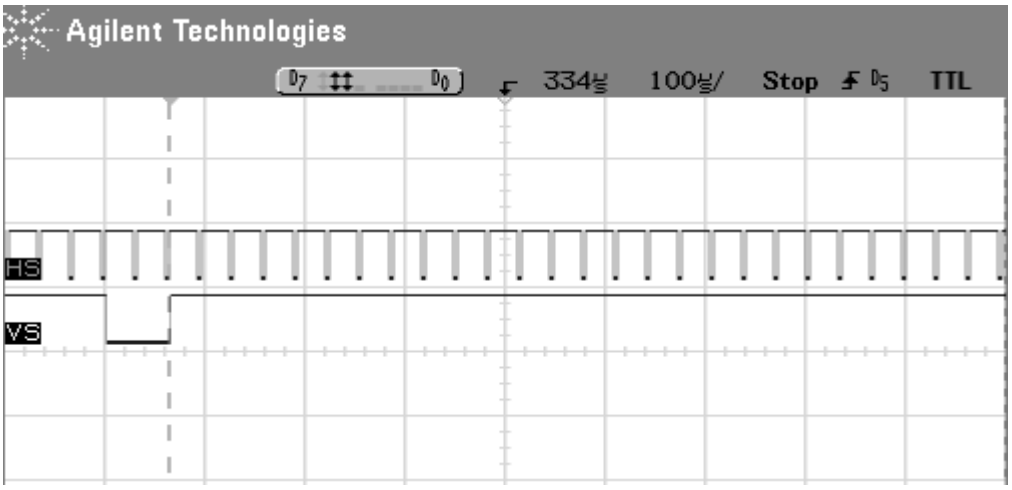


Figura 4.5 Señales de sincronización vertical y horizontal (VS y HS), generadas por el FPGA.

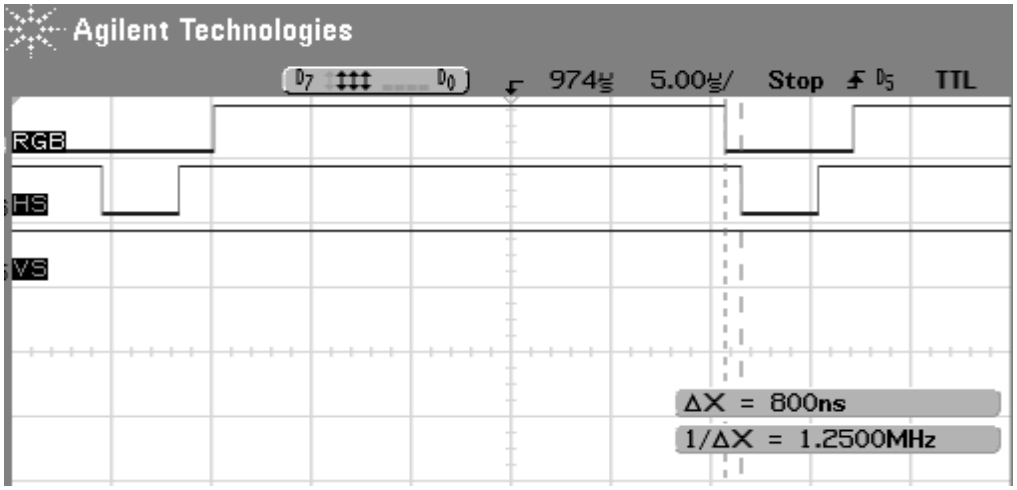


Figura 4.6 Señales de sincronización vertical y horizontal (VS, HS), además se muestra la parte donde son validos los datos a través de la línea RGB (un solo bit en "1")

Para almacenar datos en el chip SRAM, se creó un componente que acepta los datos de entrada de la PC estableciendo las señales de handshanking entre el FPGA y la tarjeta de NI, a su vez genera las señales de habilitaciones de escritura y de direcciones para ir escribiendo los datos llegantes a la SRAM. En esta prueba se envió un bloque de 16 datos los cuales se escribieron en la SRAM y luego se verificó que eran los datos correctos (ver sección 3.1.10).

4.1.4 Sistema que almacena una imagen enviada desde la PC al chip SRAM y despliegue su contenido en formato VGA

Para completar un sistema donde una imagen enviada por la PC se almacena en SRAM y se despliegue en el monitor VGA, se realizó un programa que envía los datos del componente rojo de una imagen utilizando las funciones de NIDAQ desde Matlab para establecer el handshanking de la tarjeta de NI. Se envía solo el canal rojo, ya que la biblioteca de funciones no permite hacer grupos de varios puertos, solo un puerto. Se realizó un diseño de alto nivel que unió los componentes mencionados en la sección 3.1.11. Se probó este diseño, se enviaron por medio del DAQPad-6507 260,00 datos de la imagen mencionada que se grabaron en el chip SRAM y después se desplegaron en pantalla. Se logró observar la silueta de la imagen enviada, ya que era un solo componente de color.

4.2 Resultados del diseño del sistema de imagen en el FPGA

4.3.1 Interfase a los registros de control del sensor MT9T001

Para la interfase a los registros de control del sensor MT9T001, se realizó un diseño (I2C_ESCRITURA.vhd) en el FPGA que implementa el protocolo de comunicación serial de dos hilos I2C para escribir a los registros internos del sensor MT9T001, donde el sensor se comporta como dispositivo esclavo y el FPGA como dispositivo maestro. Se utilizó una tasa de 100khz la cual es adecuada para el protocolo I2C y para los tiempos requeridos por el sensor al no estar en el máximo de frecuencia posible (ver sección 3.2.1).

Se implementaron los componentes mencionados en la sección 3.2.1, incluyendo el componente que simula la respuesta del sensor de imagen (I2C_SLAVE.vhd), se hicieron mediciones y coincide con la simulación. En la figura 4.7 se muestra una grafica del osciloscopio donde se lleva a cabo una secuencia de escritura, se observan las señales de reloj y datos (SCL, SDA) de la interfase I2C, se marcan los bits de inicio (START), paro (STOP) y de reconocimiento (ACK). Las señales ENAO, FINO y SLEN, se utilizan para indicar cuando se habilita la comunicación, cuando se termina un ciclo de escritura y cuando esta habilitado el dispositivo esclavo respectivamente.

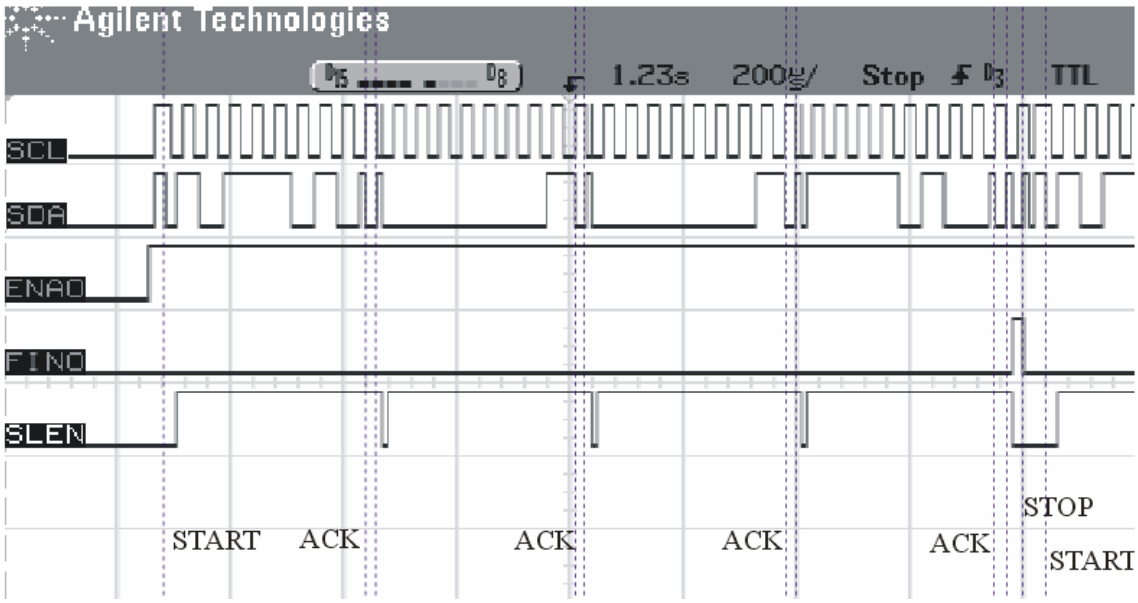


Figura 4.7 Medición de un ciclo de escritura del protocolo I2C

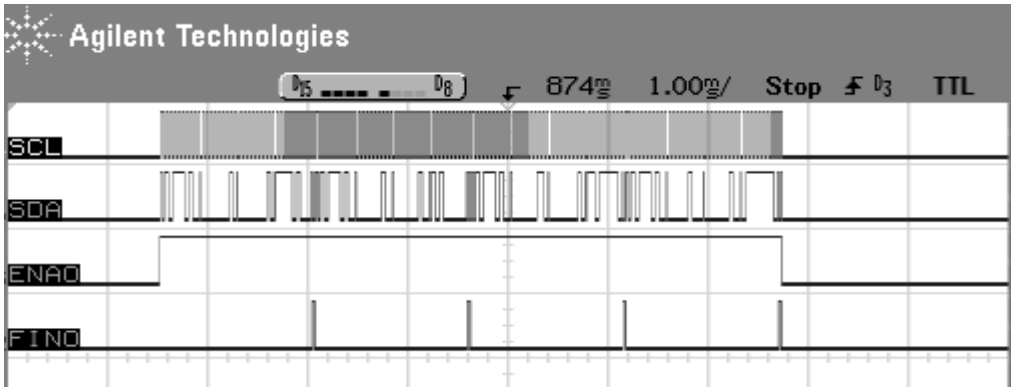


Figura 4.8 Medición de cuatro ciclos de escritura del protocolo I2C

Se hicieron mediciones de las señales del protocolo I2C con el diseño FPGA conectado al sensor MT9T001, en la figura 4.8 se muestra cuatro ciclos de escritura, donde se enviaron valores para escribir a cuatro registros, para cambiar el tamaño del cuadro (640x480) y la posición del mismo (500,672)

Dicho diseño es capaz de llevar a cabo la secuencia adecuada para escribir valores a cualquiera de los registros del sensor MT9T001, el resultado demuestra la capacidad de implementar cualquier tipo de sistema digital en un FPGA para realizar cualquier protocolo de comunicación. Además tomando en cuenta que la mayoría de los sensores CMOS de imagen modernos cuenta con interfase de comunicación serial ya sea I2C o parecida, este diseño puede ser utilizado para establecer comunicación con cualquier circuito integrado que maneje I2C.

4.3.2 Despliegue de un cuadro capturado por el sensor

Para evaluar la capacidad del sensor de capturar imagen, se realizó un diseño en el FPGA que almacena los datos de un cuadro de imagen proveniente del sensor MT9T001 en el chip de memoria SRAM de la tarjeta y lo despliega en formato VGA en un monitor estándar, como cámara "still", y se observó la imagen en tonos de verdes (ver sección 3.2.2). Con este resultado se logró utilizar el sensor MT9T001 como si fuera una cámara fotográfica, “still camera”.

4.3.3 Resultados en la obtención de los tres canales de color

Para el tratamiento de los tres componentes de color de cada píxel, se realizó un componente que implementa la arquitectura en tubería para aplicar el procesamiento de interpolación bilineal en tiempo real. Como se muestra en las figuras 3.7 y 3.9 de la sección 3.2.3. Esta arquitectura solo necesita dos buffers de línea de memoria y tres registros de tres elementos (píxeles) que forman la ventana de 3x3 de interpolación, los buffers constan de un número de elementos igual al número de píxeles de una línea del cuadro de imagen

Para esta arquitectura en tubería, se tiene una tiempo de latencia inicial $t_l = (n+2)1/f$ donde n se refiere al numero de píxeles en una línea del cuadro de imagen y f es la frecuencia del reloj, después de esta latencia se obtienen los datos interpolados un ciclo de reloj a la vez, es decir a la tasa de datos de entrada. El tiempo requerido para procesar una imagen sin latencia es $t_c = (nxm)1/f$, donde m es el número de líneas en la imagen. Para una imagen de 640x480 píxeles a una frecuencia de reloj de 48 Mhz se tiene los tiempos tl = 13.37μs y tc = 6.4ms. El tiempo global para aplicar el procesamiento de interpolación bilineal procesar a una imagen es $T = t_l + t_c$, tiempo global es T=6.4134ms

Es importante señalar que la arquitectura en tubería utilizada para aplicar la interpolación bilineal puede ser adaptada para aplicar cualquier filtro de imagen que utilice una ventana de convolución de 3 x 3, solo modificando las operaciones aritméticas necesarias pero utilizando la misma arquitectura de tubería.

4.3.4 Despliegue en tiempo real de los datos del sensor en un solo canal

Se realizó un diseño en el FPGA para llevar a cabo el despliegue en tiempo real de los datos crudos del sensor en un solo canal de color, se implementaron dos buffers de línea para capturar y desplegar en tiempo real (ver sección 3.2.4).

La condición para llevar a cabo el proceso es que el ciclo de captura debe durar el mismo tiempo que el ciclo de despliegue y además deben estar sincronizados. Se utilizó el reloj del sensor de 48 Mhz para la captura y se dividió para obtener otra señal de reloj de 24 Mhz para el despliegue en formato VGA. Al utilizar el reloj de 24 Mhz se obtiene una tasa de 53 cuadros/s para el despliegue.

Se modificaron los tiempos del sensor de imagen de tal forma que se empalmó el ciclo de captura con el de despliegue y al utilizar el mismo reloj maestro se aseguró la sincronización de los ciclos. En la figura 4.9 las líneas WREN y RDEN representan el ciclo de captura y de despliegue respectivamente, se observa que estos se encuentran sincronizados y que tienen el mismo periodo. En la figura 4.10 se muestran que el ciclo de despliegue comienza después de haber almacenado los datos de la primera línea capturada como se explicó en 3.2.5. En las figuras 4.9 y 4.10 se muestran también las líneas de sincronización de del sensor MT9T001 (FV, LV y PXCLK), validación de línea, de cuadro y reloj de píxel

Figura 4.9 Medición de los ciclos de despliegue y captura para el despliegue en tiempo real de los datos crudos del sensor, las líneas punteadas muestran el periodo de estos

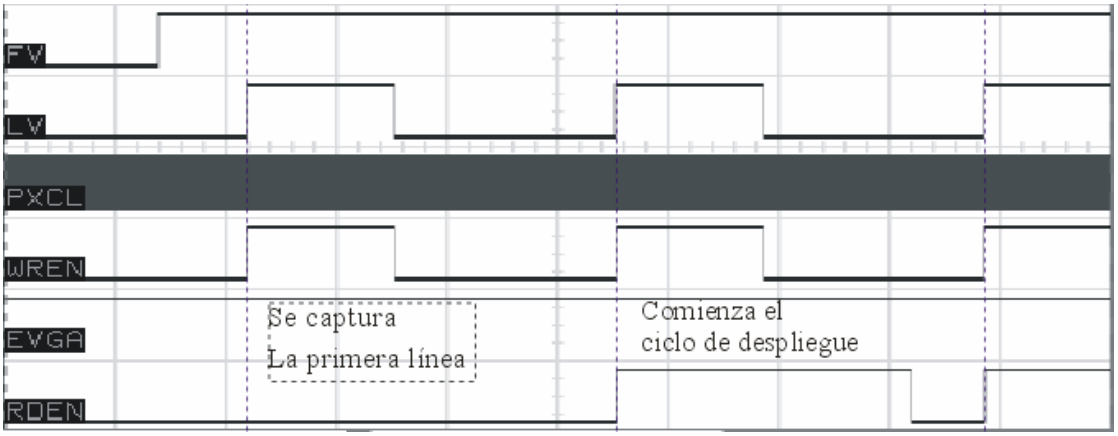


Figura 4.10 Medición de los ciclos de despliegue y captura para el despliegue en tiempo real, donde se muestra el comienzo del ciclo de despliegue

Se logró con solo dos buffers de línea llevar a cabo el proceso de despliegue y captura en tiempo real de los datos crudos del sensor, cabe mencionar que se tiene solo un latencia inicial de despliegue de $t_d = (n)1/f$, es decir solo lo que se tarda en capturar la primera línea del cuadro de imagen, donde n se refiere al numero de píxeles en una línea del cuadro de imagen y f es la frecuencia del reloj. Se observó el despliegue en tiempo real de los datos crudos del sensor en el monitor VGA en un solo canal de color (escalas de verdes). Cabe mencionar que este proceso de manejo de memoria para el despliegue en tiempo real (el uso de solo dos líneas de memoria) no se encontró en la literatura sino que fue una idea propia que se desarrollado en esta tesis.

4.3.5 Despliegue en tiempo real de los datos del sensor en los tres canales de color

Se implementó un diseño en el FPGA para hacer el despliegue en tiempo real de los datos interpolados en los 3 componentes de color, como se explica en 3.2.5 se usan tres arreglos de memoria cada uno con dos líneas de memoria.

En este diseño se tiene una latencia inicial debida al procesamiento en tubería de la interpolación bilineal más el tiempo de latencia de despliegue (tiempo de almacenar la primera línea triple de los resultados obtenidos en la interpolación).

La latencia para el despliegue en tiempo real de los datos interpolados es de

$t_r = t_l + t_d = (n+2)1/f + (n)1/f = (2n+2)1/f$

Por el procesamiento en tubería de la interpolación se tiene $t_l = (n+2)1/f$

Por el proceso de despliegue en tiempo real se tiene $t_d = (n)1/f$,

En la figura 4.11 se muestra la medición de los ciclos de captura (WREN) y despliegue (RDEN), se observa que el ciclo de despliegue comienza después de la latencia provocada por el proceso de interpolación y de haber almacenado la primera línea con los datos interpolado (por el proceso de despliegue).

Cabe mencionar que este proceso de despliegue en tiempo real de los datos interpolados, se lleva a cabo utilizando solamente dos arreglos de memoria con dos líneas de memoria cada uno, es decir, con solo seis líneas de memoria del cuadro de imagen, por lo que el diseño utilizado cumple a sobre manera las restricciones de memoria de la gran mayoría de los dispositivos FPGA.

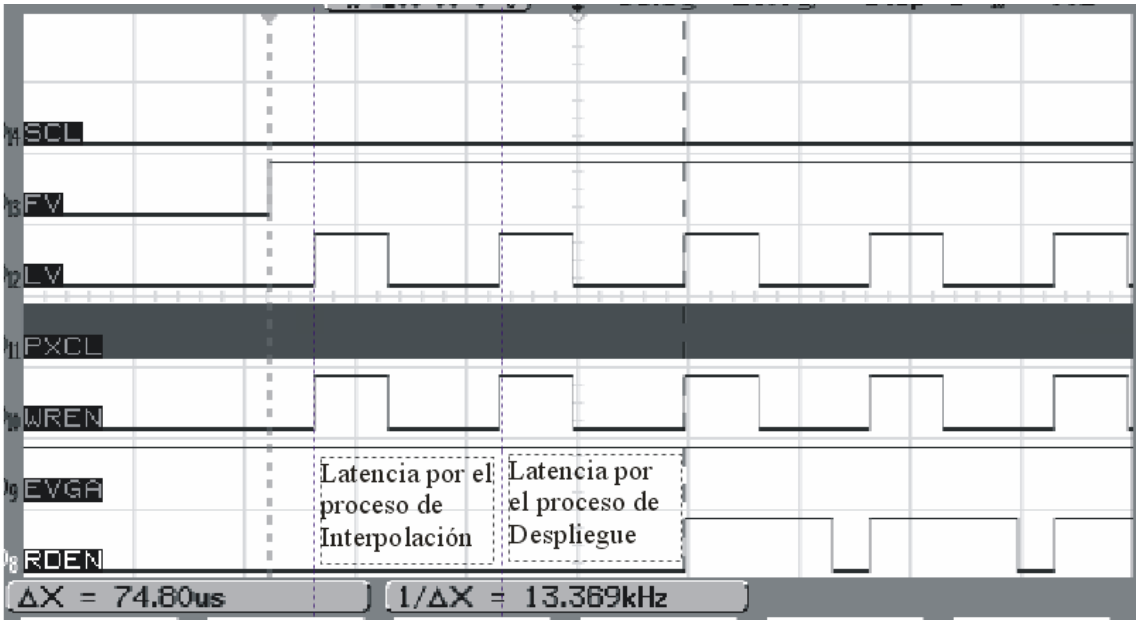


Figura 4.11 Medición de los ciclos de captura y despliegue para el proceso de despliegue en tiempo real de los datos interpolado en los tres canales de color

4.3.6 Resultados de la corrección de color

Se realizó un componente que forma la arquitectura necesaria para aplicar la corrección de color a los datos interpolados, la matriz de corrección se implementa a través de la suma de los productos de los datos interpolados por los coeficientes de la matriz de corrección. Como se explicó en 3.2.6 se aplican los productos utilizando los multiplicadores embebidos del FPGA cyclone II, por lo que se implementan en paralelo, solo se tiene una latencia inicial $t_m = 1/f$, de un ciclo de reloj debido a que el resultado del producto se condiciona a un valor máximo o a cero y se ocupa ponerlo a registro que se actualiza cada ciclo de reloj. Cabe mencionar que el componente utilizado arma el esqueleto necesario para aplicar la matriz de corrección y que se pueden editar dentro del código distintos valores para los coeficientes de la matriz.

Los coeficientes de la matriz de corrección deben encontrarse para cada caso particular, el encontrar los valores de los coeficientes requiere de un proceso de calibración donde se utiliza un patrón de color. Cabe mencionar que la búsqueda de estos coeficientes no es una tarea trivial y que incluso compañías fabricantes de sensores de imagen ofrecen este servicio, un ejemplo de esto se puede observar en [18].

4.3.7 Sistema completo que aplica la configuración del sensor, la interpolación bilineal, la corrección de color y el despliegue en tiempo real en los 3 canales de color

Se creó el sistema completo (DESP_BAYER3.vhd) que une los componentes para llevar a cabo la configuración del sensor (I2C_ESCRITURA.vhd), la interpolación bilineal (PIPEBILINEAL.vhd), la corrección de color (MUTLADD5.vhd) y el despliegue en tiempo real de los 3 canales de color (DESP_REAL4.vhd). En este sistema el tiempo de latencia inicial t_r, es la suma de las latencias provocadas por el procesamiento de interpolación t_l, el proceso de despliegue en tiempo real t_d y el provocado por aplicar la corrección de color t_m. Después de este tiempo de latencia t_r se empiezan a desplegar imagen del sensor en formato VGA, el tiempo de inicial t_r se muestra en la ecuación 4.1:

4.1 $$t_r = t_l + t_d + t_m = (n+2)1/f + (n)1/f + 1/f = (2n+3)1/f$$

Cabe mencionar que se creó una paquete[14] (PAQUETE.vhd) con las declaraciones de todos los componentes (componentes comunes que puede ser utilizados por varios diseños), también se creó un paquete (CTES.vhd) donde se guardan parámetros utilizados por los componentes del sistema de tal forma se pueden configurar los valores del numero de píxeles en un línea, números de bits de cada píxel. También se puede configurar la arquitectura de procesamiento en tubería en cuanto el número de buffers de línea, el número de registros para la ventana del filtro. Estos paquetes (PAQUETE.vhd y CTES.vhd) se muestran en el apéndice C.

En la figura 4.12 se muestra un diagrama a bloques del sistema completo que aplica la configuración del sensor, la interpolación bilineal, la corrección de color y el despliegue en tiempo real en los 3 canales de color, este sistema fue implementado completamente en el dispositivo FPGA Cyclone II.

[14] Un paquete en VHDL se utiliza para almacenar especificaciones usadas frecuentemente tales como tipos de datos, declaraciones de componentes, declaraciones de constantes entre otros. Los elementos definidos en un paquete se pueden utilizar por cualquier entidad

Los recursos utilizados del FPGA Cyclone II para implementar el sistema completo DESP_BAYER3.vhd son los siguientes:

Total de elementos lógicos 930/33,216 (2.8%)

Total de bits de Memoria 51,160/483,840 (11%)

Multiplicadores embebidos de 9 bits 10/70 (14%)

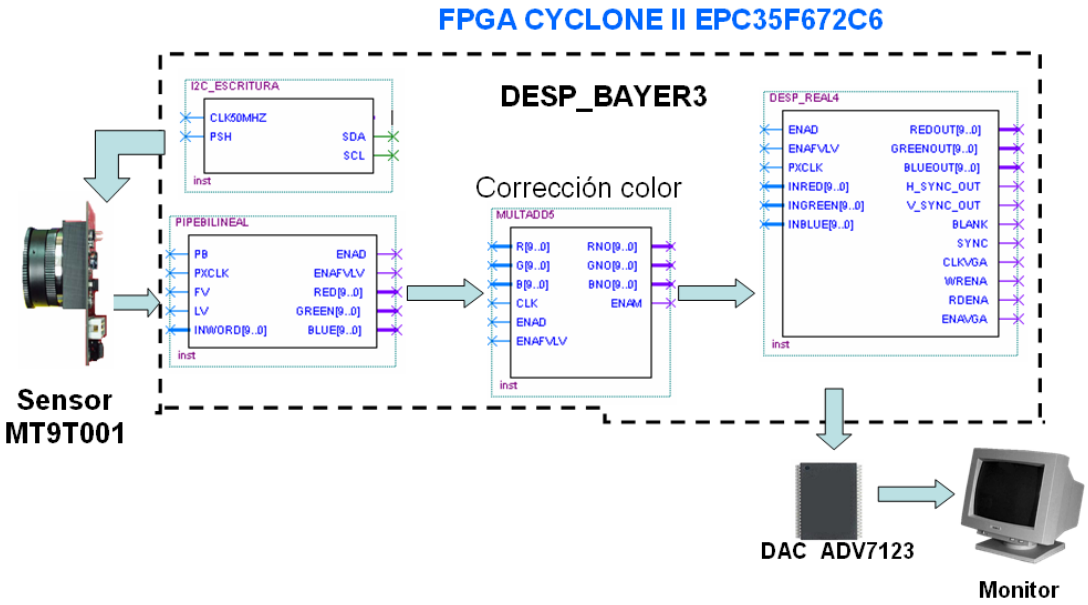


Figura 4.12. Sistema completo que aplica la configuración del sensor, la interpolación bilineal, la corrección de color y el despliegue en tiempo real en los 3 canales de color

Capítulo V

Conclusiones y trabajo futuro

5.1 Conclusiones

Se logró formar un ambiente de desarrollo para diseño de sistemas digitales en FPGA, para la captura, procesamiento y despliegue de imagen, este producto de cámara inteligente que puede ser utilizado en aplicaciones que requieran procesamiento de video a bajo costo y en aplicaciones de sistema de visión que requieren de portabilidad, ya que es un sistema embebido.

El sistema de cámara inteligente desarrollado incrementa la velocidad para el procesamiento de imagen de operaciones de vecindad ya que los algoritmos se implementan en hardware, lo que permite obtener resultados a la tasa de entrada del píxel después de una latencia inicial requerida para almacenar dos líneas de píxeles del cuadro (no se necesita esperar que se almacene un cuadro completo para obtener resultados). Debido a esto este sistema se puede utilizar para aplicaciones de sistema de visión basado en software, como un subsistema que acelere el procesamiento de las operaciones de imagen que requieren gran cantidad de cálculos y dejar al basado en software a cargo de otro tipo de tareas.

Se puede utilizar el mismo reloj (tasa de píxel) del sensor de imagen, es decir, no se requiere de utilizar reloj de alta velocidad como los microprocesadores dedicados o de propósito general

Gracias a la utilización del sensor CMOS inteligente (permite ser programado) en el sistema desarrollado, se puede modificar los tamaños de cuadros, tasas de cuadro, lo que permite obtener distintos formatos haciendo mas versátil las posibles aplicaciones de este producto

Los recursos utilizados por el diseño completo del sistema implementado en esta tesis son de un porcentaje mínimo de los recursos del FPGA utilizado (ver sección 4.3.7), por lo cual se puede utilizar los recursos disponibles, para implementar otras arquitecturas de procesamiento para realizar otras funciones de imagen, para integrar

otro tipo de conectividad al sistema así como otros funciones que la diferencien de otros productos

5.2 Trabajo futuro

En este proyecto de tesis, se logró formar una plataforma de trabajo para diseño de sistemas digitales en FPGA. Se puede mejorar a la velocidad de transferencia de datos desde la PC, utilizando otro dispositivo de E/S digitales, como se mencionó en la sección 4.1.2. De esta forma la verificación de los algoritmos implementados en el FPGA por medio de la PC se puede llevar a cabo

La metodología de diseño jerárquico utilizada en esta tesis por medio del lenguaje VHDL, permite reutilizar las componentes ya existentes y crear nuevos, por lo que se desea utilizar la arquitectura en tubería implementada para la interpolación bilineal, para aplicar otras funciones de imagen de bajo nivel que utilizan operaciones de vecindad (tienen la misma estructura), tales como los filtros lineales entre los que se pueden mencionar: detectores de bordes, los filtros promediadores, filtros para realce, entre otros.

La arquitectura utilizada para implementar el despliegue en tiempo real, hace un manejo de memoria novedoso, en el que se utilizan tan sólo unas cuantas líneas de memoria, por lo que se piensa que es producto de publicación

En cuanto al componente implementado para aplicar la matriz de corrección de color, quedó pendiente encontrar los coeficientes adecuados, por lo que se desea utilizar alguna metodología existente para la búsqueda de los mismos, con lo que se obtendrían imágenes en color que se perciban adecuadamente por el ser humano

Se desea crear una circuito impreso (tarjeta personalizada), donde se conecten todos los elementos utilizados en esta tesis (Sensor de imagen, FPGA, DAC de video, chip de SRAM, oscilador), de tal forma que se aplique la cámara inteligente para aplicaciones que requieran de movilidad.

[1] Yu Shi, Parnesh Raniga, Ismail Mohamed. A Smart Camera for Multimodal Human Computer Interaction. 1-4244-0216-6/06/$20.00 ©2006 IEEE.

[2] Caarls W., Jonker P., Corporaal H. Smart Cam: Devices for Embedded Intelligent Cameras. 3RD PROGRESS workshop on Embedded Systems. Mariel Schweizer. 2002.

[3] Wayne Wolf, Burak Ozer, Tiehan Lv. Smart Cameras as Embedded Systems. IEEE Computer. 35. 9. 48–53. 2002.

[4] Harry Broers, Wouter Caarls, Pieter Jonker, Richard Kleihorst. Architecture Study for Smart Cameras. Proc. EOS Conference on Industrial Imaging and Machine Vision (Munich, Germany, June 13-15), European Optical Society, 2005, 39-49.

[5] El Gamal A., Eltoukhy H., CMOS Image Sensors, IEEE Circuits & Devices Magazine May/June 2005

[6] 1/2-Inch 3-Megapixel CMOS Digital Image Sensor MT9T001P12STC, Micron Technology, Inc, 2004

[7] Rastislav Lukac et al. A New CFA interpolation framework. Signal Processing 86 (2006) 1559–1579

[8] Ting Chen. A Study of Spatial Color Interpolation Algorithms for single-Detector digital Cameras. Stanford University. http://scien.stanford.edu/class/psych221/projects/99/tingchen/index.htm

[9] Sharma Ashok K., Programable Logic Handbook PLDs, CPLDs & FPGAs. McGraw-Hill . Primera Edición, New York. Pp. 435 1998

[10] Maxinez D., Alcalá J. VHDL El arte de programar sistemas digitales. Compañía editorial continental. Primera Edición. México. pp 352. 2002

[11] Salcic Z., Smailagic A. Digital System Design and prototyping Using Field Programmable Logic and Hardware Description Languages. Kluwer Academic Publishers. Segunda edición. Boston. pp 620. 2000.

[12] Hamblen J., Furman M.. Rapid Prototyping of Digital Systems A Tutorial Approach. Kluwer Academic Publishers. Segunda edición. Boston. pp 270. 2001.

[13] The Data Acquisition Toolbox for use with MATLAB, User's Guide. The MathWorks,Inc. Marzo 2005

[14] NI-DAQ MATLAB toolbox, página web: http://www.alamath.com/index.php?option= com_content&task=view&id=16&Itemid=1

[15] NI Discussion Forums: Most Active Hardware Boards: Digital I/O: DaqPad 6507 DIO speed limit, pagina web: http://forums.ni.com/ni/board/message?board.id=70&message.id=5328.

[16] Chi-Jeng Chang, Zen-Yi Huang, Hsin-Yen Li, Kai-Ting Hu, and Wen-Chih Tseng. Pipelined Operation of Image Capturing and Processing. Proceedings of 2005 5th IEEE Conference on Nanotechnology, Nagoya, Japan, Julio 2005

[17] Quartus II Handbook,Volume 1, 7 Recommended HDL Coding Styles. Altera Corporation. Marzo 2007

[18] Color Correction for Image Sensors Application Notes. Image Sensor Solutions. Kodak. Octubre, 2003

[19] Color Correction Matrix Application Note. Lumenera Corporation, 2005

[20] DAQPad-6507/6508 User Manual Digital I/O Devices for USB, National Instruments Corporation. 1998.

[21] M Series User Manual. National Instruments Corporation, Junio 2007

Apéndice A Códigos en VHDL, programas en Matlab y en Visual C++, realizados de la sección 3.1.1 a 3.1.11

"**SERIAL.vhd**"

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY Serial IS
                PORT
        (       LINE_VALID, PIXCLK    :IN    STD_LOGIC;
                DATAIN                :IN    STD_LOGIC_VECTOR(1 DOWNTO 0);
                CLKOUT                :OUT   STD_LOGIC;
                DATAOUT               :OUT   STD_LOGIC_VECTOR(1 DOWNTO 0));

END Serial;

ARCHITECTURE a OF Serial IS
        SIGNAL DATA : STD_LOGIC_VECTOR(1 DOWNTO 0);
        SIGNAL CLK : STD_LOGIC;
BEGIN

CLK<=PIXCLK;
CLKOUT<=CLK;

PROCESS
BEGIN
WAIT UNTIL(PIXCLK'EVENT)AND(PIXCLK= '0');
 IF (LINE_VALID='1') THEN
 DATA<=DATAIN;
 DATAOUT<=DATA;
 ELSE
 DATAOUT<="00";
 END IF;
END PROCESS ;
END a;
```

“RAM.vhd”

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY RAM IS
                PORT
        (       LINE_VALID,PIXCLK   :IN     STD_LOGIC;
                DATAIN              :IN     STD_LOGIC_VECTOR(1 DOWNTO 0);
                CLKOUT              :OUT    STD_LOGIC;
                DATAOUT             :OUT    STD_LOGIC_VECTOR(1 DOWNTO 0));
END RAM;
ARCHITECTURE a OF RAM IS
        SIGNAL DIR  : INTEGER RANGE 0 TO 3:=0 ;
        TYPE RAM_TYPE IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(1 DOWNTO 0);
        SIGNAL TRAM : RAM_TYPE;
BEGIN
CLKOUT<=PIXCLK;

SUBIDA:PROCESS
BEGIN
WAIT UNTIL(PIXCLK'EVENT)AND(PIXCLK= '1');
IF (LINE_VALID='1') THEN
  DATAOUT<=TRAM(DIR);
         IF (DIR=3)THEN
         DIR<=0;
         ELSE
         DIR<=DIR+1;
         END IF;
ELSE
DATAOUT<="00";
END IF;
END PROCESS SUBIDA ;

BAJADA:PROCESS
BEGIN
WAIT UNTIL(PIXCLK'EVENT)AND(PIXCLK= '0');
IF (LINE_VALID='1') THEN
TRAM(DIR)<=DATAIN;
ELSE
TRAM(DIR)<="00";
END IF;
END PROCESS BAJADA ;
END a;
```

“Prueba.m”

```
clear
clc
dio = digitalio('nidaq', 1);
hline = addline(dio, 0:5, 'Out');
t=0.01;
A =[3 2 3 1;2 1 2 1 ;2 1 3 1];
[m,n] = size(A);
   putvalue(dio,0)
for  i= 1:m,
   putvalue(dio.Line(1), [1])
   pause(t/2)
   for j = 1:n,
            putvalue(dio.Line(2), [1])
            pause(t/2)
            putvalue(dio.Line(3:4), [A(i,j)])
            pause(t/2)
            putvalue(dio.Line(2), [0])
            pause(t/2)
            putvalue(dio.Line(3:4), [0 0])
            pause(t/2)
   end
   putvalue(dio.Line(1),[0])
   pause(t*2)
end
   putvalue(dio.Line(2), [1])
   pause(t*2)
   putvalue(dio.Line(2), [0])
   pause(t*2)
   putvalue(dio.Line(2), [1])
   pause(t*2)
   putvalue(dio.Line(2), [0])
   pause(t*2)
for i=1:4
 putvalue(dio.Line(2), [1])
 pause(t)
 putvalue(dio.Line(2), [0])
 pause(t)
 putvalue(dio.Line(2), [1])
 pause(t)
 putvalue(dio.Line(2), [0])
 pause(t)
end
 end
```

“ENVIAR.vhd”

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY ENVIAR IS
                PORT
        (       CLK25,AVISO     :IN     STD_LOGIC;
                ENV             :OUT    STD_LOGIC;
                CLKOUT          :BUFFER STD_LOGIC;
                DATARE          :OUT    STD_LOGIC_VECTOR(1 DOWNTO 0));
END ENVIAR;

ARCHITECTURE a OF ENVIAR IS
        SIGNAL INDX  : INTEGER range 0 to 13:=0 ;
        SIGNAL CONT : INTEGER :=0 ;
        TYPE RAM_TYPE IS ARRAY (0 TO 13) OF STD_LOGIC_VECTOR(1 DOWNTO 0);
        SIGNAL TRAM : RAM_TYPE;
BEGIN

ENVIADA:PROCESS
BEGIN
        WAIT UNTIL(CLKOUT'EVENT)AND(CLKOUT='0');
IF (NOT AVISO='1') THEN
 DATARE<=TRAM(INDX);
 ENV<='1';
        IF(INDX=13)THEN
    ENV<='0';
        END IF;
ELSE
 DATARE<="00";
END IF;
        END PROCESS ENVIADA;

ENVIADA2:PROCESS
BEGIN
        WAIT UNTIL(CLKOUT'EVENT)AND(CLKOUT='1');
IF (NOT AVISO='1') THEN
  IF (INDX=13)THEN
         INDX<=0;
         ELSE
         INDX<=INDX+1;
         END IF;
END IF;
        END PROCESS ENVIADA2;

CLK25aCLKOUT:PROCESS
BEGIN
WAIT UNTIL(CLK25'EVENT)AND(CLK25= '1');
        IF (CONT>=125)THEN
        CLKOUT<='0';
                IF (CONT=249)THEN
                CONT<=0;
                ELSE
                CONT<=CONT+1;
                END IF;
        ELSE
```

“ENVIAR.vhd” Continuación

```
   CLKOUT<='1';
          CONT<=CONT+1;
          END IF;
END PROCESS CLK25aCLKOUT ;

TRAM(0)<="11";
TRAM(1)<="11";
TRAM(2)<="10";
TRAM(3)<="11";
TRAM(4)<="01";
TRAM(5)<="10";
TRAM(6)<="11";
TRAM(7)<="10";
TRAM(8)<="11";
TRAM(9)<="10";
TRAM(10)<="01";
TRAM(11)<="11";
TRAM(12)<="01";
TRAM(13)<="00";

END a;
```

“Enviar.m”

```
clear
clc

status=DIG_Prt_Config(1,0,0,1);
status=DIG_Prt_Config(1,1,0,1);
t=0.01;
A =[3 2 3 1;2 3 2 3 ;2 1 3 1];
[m,n] = size(A);

   status = DIG_Out_Port (1,0,0000);
   status = DIG_Out_Port (1,1,0000);

for i=1:m,
   status=DIG_Out_Line (1, 1, 2, 1); % Valid (bit 2 puerto 1=1)
   pause(t/2)

   for j = 1:n,
            status=DIG_Out_Line (1, 1, 3, 1);% clk (bit 3 puerto 1=1)
            pause(t/2)
            status=DIG_Out_Port (1,0,A(i,j));% Pone  A(i,j) en los bits del puerto 0 (datos)
            pause(t/2)
            status=DIG_Out_Line (1, 1, 3, 0);% clk (bit 3 puerto 0=0)
            pause(t/2)
            status=DIG_Out_Port (1,0,0000);% Pone en 0 los bits del puerto 0 (datos)
            pause(t/2)

   end

   status=DIG_Out_Line (1, 1, 2, 0); % Valid (bit 2 puerto 1=0)
   pause(t*2)

end
   status=DIG_Out_Line (1, 1, 3, 1); % clk (bit 3 puerto 1=1)
   pause(t*2)
   status=DIG_Out_Line (1, 1, 3, 0); % clk (bit 3 puerto 1=0)
   pause(t*2)
   status=DIG_Out_Line (1, 1, 3, 1); % clk (bit 3 puerto 1=1)
   pause(t*2)
   status=DIG_Out_Line (1, 1, 3, 0); % clk (bit 3 puerto 1=0)
   pause(t*2)
```

“Recibir.vhd”

```
clear
clc

A =[3 2 3 1;2 3 2 3 ;2 1 3 1];
[m,n] = size(A);
t=0.001;
k=1;
l=1;
ibfa=0;
ena=1;

status=DIG_Prt_Config(1,0,1,0); % configurar como entrada handshaked al puerto 0
status=DIG_Prt_Config(1,3,0,1); % configurar como salida no handshk al puerto 3
status=DIG_Out_Line (1,3,0,0);  %poner en 0 el bit 0 del puerto 3 (Para el FPGA es aviso=0)

%--------------------------------------------------
while ena==1 %mientras ena=1 hará el ciclo; ena sería el equivalente de aviso
   if ena==0
     break
   end
%--------------------------------------------------
status=DIG_Out_Line (1,3,0,1);  %poner en 1 el bit 0 del puerto 3 (Para el FPGA es aviso=1)
pause(t)
status=DIG_Out_Line (1,3,0,0);  %poner en 0 el bit 0 del puerto 3 (Para el FPGA es aviso=0)

while ibfa==0 %Se espera hasta que este listo para recibir
   [status,ibfa]= DIG_Prt_Status (1,0);%Checar si el puerto 0 esta listo para recibir(ibfa=1)
   if ibfa==1
      continue
      end
   end

%PROCESO
%-----------------------------------------------
[status,B(k,l)] = DIG_In_Port (1,0); %Ingresar datos desde el puerto 0 a la matriz B
   ibfa=0;   %limpiar la variable para hacer la siguiente iteracion (se require ibfa=0 para el while del Ptr
status)
   if k>m
   ena=0;
   else
       if l==n
      l=1;
      k=k+1;
       else
      l=l+1;
       end
   end
%--------------------------------------------------
pause(t)
end
  B
```

“Enviarblock.m”

```
clc
clear

% Estas instrucciones son usadas para sacar un arreglo de 4 datos llamado
% bufferout, (estas son usadas para sacar la imagen, solo que se carga al

status = DIG_SCAN_Setup (1,1,0,1);
bufferout=[0;1;0;1;0];
a= DIG_Block_Out(1,1,bufferout,4);
```

“Recibirblock.m”

```
clc
clear

% Estas instrucciones son para recibir 4 datos iguardarlos en un arreglo
% llamado bufferin
status = DIG_SCAN_Setup (1,1,0,0);
[b, bufferin] = DIG_Block_In(1,1,4)
[c, remain] = DIG_Block_Check(1,1)
d= DIG_Block_Clear(1,1)
```

“ENVIAR02.vhd”

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY ENVIAR02 IS
        PORT
        (IBF,CLK: IN    STD_LOGIC;
         DATAOUT: OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
         STB:    OUT    STD_LOGIC);
END ENVIAR02;

ARCHITECTURE a OF ENVIAR02 IS
  TYPE TIPOESTADO IS (EDOA,EDOB);
        SIGNAL ESTADO:TIPOESTADO;
        SIGNAL CONT,CONT1,CONT2:INTEGER:=0 ;
        TYPE RAM_TYPE IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(1 DOWNTO 0);
        SIGNAL TRAM : RAM_TYPE;
BEGIN

PROCESS (CLK)
BEGIN
IF CLK'EVENT AND CLK='1' THEN
CASE ESTADO IS
        WHEN EDOA =>
IF IBF='0' THEN
        IF CONT>3 THEN-------PARA REINICIAR EL ARREGLO EN LA PRIMERA LOCALIDAD
        CONT<=0;
        END IF;
        IF CONT1=25000 THEN ----------PARA HACER EL DELAY DE IBF=0 A STB=0
        CONT1<=0;
     ESTADO<=EDOB;
     DATAOUT<=TRAM(CONT);--SACAR LO QUE HAY EN DICHA LOCALIDAD DEL
ARREGLO
     ELSE
     CONT1<=CONT1+1;
     END IF;
  ELSE
  ESTADO<=EDOA;
END IF;
        WHEN EDOB =>
        IF IBF='0' OR IBF='1' THEN
                IF CONT2=25000 THEN----------PARA HACER EL ANCHO DE STB
                CONT2<=0;
                CONT<=CONT+1;-------PARA CAMBIAR LA LOCALIDAD DEL ARREGLO
                ESTADO<=EDOA;
                ELSE
                CONT2<=CONT2+1;
                ESTADO<=EDOB;
                END IF;
        END IF;
END CASE;
END IF;
END PROCESS;

WITH ESTADO SELECT
        STB<='1' WHEN EDOA,
                 '0' WHEN EDOB;
```

“ENVIAR02.vhd” Continuación

```
TRAM(0)<="01";
TRAM(1)<="10";
TRAM(2)<="11";
TRAM(3)<="00";
END a;
```

“RECIBIR01.vhd”

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY RECIBIR01 IS
        PORT
        (OBF,CLK:  IN   STD_LOGIC;
         ACK: OUT       STD_LOGIC);
END RECIBIR01;
ARCHITECTURE a OF RECIBIR01 IS
  TYPE TIPOESTADO IS (EDOA,EDOB);
        SIGNAL ESTADO:TIPOESTADO;
        SIGNAL CONT1,CONT2:INTEGER:=0 ;

BEGIN

PROCESS (CLK)
BEGIN
IF CLK'EVENT AND CLK='1' THEN
CASE ESTADO IS
        WHEN EDOA =>
        IF OBF='0' THEN
        IF CONT1=50000 THEN
        CONT1<=0;
     ESTADO<=EDOB;
     ELSE
     CONT1<=CONT1+1;
     END IF;
  ELSE
  ESTADO<=EDOA;
        END IF;

        WHEN EDOB =>
        IF OBF='0' OR OBF='1' THEN
                IF CONT2=50000 THEN         --------hacer el ancho de ACK
                CONT2<=0;
                ESTADO<=EDOA;
                ELSE
                CONT2<=CONT2+1;
                ESTADO<=EDOB;
                END IF;
        END IF;

END CASE;
END IF;
END PROCESS;

WITH ESTADO SELECT
        ACK<='1' WHEN EDOA,
                 '0' WHEN EDOB;
END a;
```

“DIsingleBufHandshake2.C”

```
#include "nidaqex.h"
void main(void)
{
   / * Declaracion de variables locales:*/
   i16 iStatus = 0;
   i16 iRetVal = 0;
   i16 iDevice = 1;
   i16 iGroup = 1;
   i16 iGroupSize = 1;
   static i16 piPortList[1] = {0};
   i16 iDir = 0;
   static i16 piBuffer[4] = {0};
   u32 ulCount = 4;
   u32 ulRemaining = 1;
   i16 iIgnoreWarning = 0;
   i16 iYieldON = 1;

   /* Se configura un grupo de un puerto como entrada, con handshaking.   */

   iStatus = DIG_SCAN_Setup(iDevice, iGroup, iGroupSize, piPortList,
    iDir);
   iRetVal = NIDAQErrorHandler(iStatus, "DIG_SCAN_Setup",
    iIgnoreWarning);

   /* Inicia la trasnferencia de entrada de 4 "datos". el tamaño de piBuffer es ulCount /

   iStatus = DIG_Block_In(iDevice, iGroup, piBuffer, ulCount);
   iRetVal = NIDAQErrorHandler(iStatus, "DIG_Block_In",
    iIgnoreWarning);

  printf(" Apply your handshaking signals to the appropriate handshaking I/O pins.\n");

   while ((ulRemaining != 0) && (iStatus == 0)) {
      iStatus = DIG_Block_Check(iDevice, iGroup, &ulRemaining);
      iRetVal = NIDAQYield(iYieldON);   }
   iRetVal = NIDAQErrorHandler(iStatus, "DIG_Block_Check",
    iIgnoreWarning);

   /* limpia la operacion de block. */
   iStatus = DIG_Block_Clear(iDevice, iGroup);

   /* Desconfigura el grupo. */
   iStatus = DIG_SCAN_Setup(iDevice, iGroup, 0, piPortList, iDir);
   iStatus = NIDAQPlotWaveform(piBuffer, ulCount, WFM_DATA_I16);

 printf(" The data is available in 'piBuffer'.\n");
   if (iStatus == 0) {
      printf(" Digital handshaked buffered input is done!\n");
   }
}
```

“MEMFILE.vhd”

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY MEMFILE IS
        PORT
        (SRAM_ADDR          :OUT STD_LOGIC_VECTOR(17 DOWNTO 0) ;
         SRAM_DQ            :INOUT STD_LOGIC_VECTOR(15 DOWNTO 0);
         SWADD              :IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         SWDAT              :IN STD_LOGIC_VECTOR(1 DOWNTO 0);
         LEDADD             :OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
         LEDDAT             :BUFFER STD_LOGIC_VECTOR(1 DOWNTO 0);
         LEDWE              :OUT STD_LOGIC; --LEDCLK
        SWWE                :IN STD_LOGIC;
         WE                 :OUT STD_LOGIC;
         CE,OE,UB,LB         :OUT STD_LOGIC);
END MEMFILE;


ARCHITECTURE a OF MEMFILE IS
        SIGNAL ADDR :STD_LOGIC_VECTOR(3 DOWNTO 0);
        SIGNAL WRITE,CL :STD_LOGIC;
BEGIN
--Poner las entradas de control del chip SRAM S61LV25616AL a 0 para modo sencillo, controlando solo
----con WE

CE<='0';OE<='0';UB<='0';LB<='0';
SRAM_ADDR(17 DOWNTO 4)<="00000000000000";  --Direcciones que no se utilizaran del chip
SRAM
ADDR<=SWADD;
LEDADD<=ADDR;
SRAM_ADDR(3 DOWNTO 0)<=ADDR;
WRITE<=SWWE;
WE<=WRITE;
LEDWE<=WRITE;
LEDDAT<=SRAM_DQ(1 DOWNTO 0);

SRAM_DQ(1 DOWNTO 0)<=SWDAT WHEN WRITE='0' ELSE "ZZ";

END a;
```

“VGA_SYNC.vhd”

```
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.PAQUETE.ALL;

ENTITY VGA_SYNC IS
        PORT
        ( CLK50MHZ                              : IN STD_LOGIC;
          RED,GREEN,BLUE                        : BUFFER STD_LOGIC_VECTOR(9 DOWNTO
0);
          REDOUT,GREENOUT,BLUEOUT       : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
          H_SYNC_OUT,V_SYNC_OUT,CLK25MHZ        : OUT STD_LOGIC;
          PIXEL_ROW,PIXEL_COLUMN                : OUT STD_LOGIC_VECTOR(9
DOWNTO 0));
END VGA_SYNC;

ARCHITECTURE a OF VGA_SYNC IS
        SIGNAL CLKWIRE                  :STD_LOGIC;
        SIGNAL H_SYNC,V_SYNC        :STD_LOGIC;
        SIGNAL VIDEO_ON,VIDEO_ON_V,VIDEO_ON_H :STD_LOGIC_VECTOR(9 DOWNTO
0);
        SIGNAL H_COUNT,V_COUNT          :STD_LOGIC_VECTOR(9 DOWNTO 0);

--Se utiliza un componente que divida la frecuencia del reloj de 50mhz a 25mhz
BEGIN
ET :CLKDIVIDE PORT MAP(CLK50MHZ=>CLK50MHZ,CLK25MHZ=>CLKWIRE);
CLK25MHZ<=CLKWIRE;

VIDEO_ON<=VIDEO_ON_H AND VIDEO_ON_V;
RED<="1111111111";
GREEN<="1111111111";
BLUE<="1111111111";

PROCESS
BEGIN
        WAIT UNTIL (CLKWIRE'EVENT) AND (CLKWIRE='1');
                --Generar las señales de tiempos Horizontal y Vertical para la señal de video
                --H_COUNT cuenta píxeles(640+tiempo extra para señales de sincronizacion)
                --
                --HORIZ_SYNC ----------_____-----
                --H_COUNT       0       640       659    755   799
IF (H_COUNT=799) THEN
H_COUNT<="0000000000";
ELSE
H_COUNT<=H_COUNT+1;
END IF;
                --Generar Señal de sincronizacion Horizontal usando H_COUNT--
IF (H_COUNT<=755) AND (H_COUNT>=659) THEN
        H_SYNC<='0';
ELSE
   H_SYNC<='1';
END IF;
```

"VGA_SYNC.vhd" Continuación

```
--V_COUNT cuenta renglones de píxeles (480+ tiempo extra para señales de sincronizacion)--
                                    --VERT_SYNC ------------------------_______---------
                                    --V_COUNT       0       480       493-494    524

IF (V_COUNT>=524) AND (H_COUNT>=799) THEN
   V_COUNT<="0000000000";
ELSIF (H_COUNT=799) THEN
        V_COUNT<=V_COUNT+1;
END IF;

                                    --Generar señal de sincronizacion Vertical usando V_COUNT
IF (V_COUNT<=494) AND (V_COUNT>=493) THEN
        V_SYNC<='0';
ELSE
        V_SYNC<='1';
END IF;

                                    --Generar Señales de Video en Pantalla para los datos de los píxeles
IF (H_COUNT<=639) THEN
        VIDEO_ON_H<="1111111111";
        PIXEL_COLUMN<=H_COUNT;
ELSE
        VIDEO_ON_H<="0000000000";
END IF;


IF (V_COUNT<=479) THEN
   VIDEO_ON_V<="1111111111";
   PIXEL_ROW <=V_COUNT;
ELSE
        VIDEO_ON_V<="0000000000";
END IF;

                        --Se pasan todas las señales de video a través de DFFs para eliminar
                        --cualquier retardo que cause una imagen borrosa
                        --Apaga las salidas RGB cuando estén fuera del área de despliegue de video
REDOUT   <=RED AND VIDEO_ON;
GREENOUT <=GREEN AND VIDEO_ON;
BLUEOUT  <=BLUE AND VIDEO_ON;
H_SYNC_OUT<=H_SYNC;
V_SYNC_OUT<=V_SYNC;

END PROCESS;
END a;
```

“CLKDIVIDE.vhd”

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY CLKDIVIDE IS
                    PORT
          (CLK50MHZ   :IN STD_LOGIC;
          CLK25MHZ    :OUT  STD_LOGIC;
          CLK1MHZ     :OUT STD_LOGIC;
          CLK100KHZ  :OUT STD_LOGIC);
END CLKDIVIDE;
ARCHITECTURE a OF CLKDIVIDE IS
          SIGNAL CONT,CONT2,CONT3: INTEGER:=0 ;
BEGIN
RELOJ25MHZ:PROCESS
BEGIN
WAIT UNTIL(CLK50MHZ'EVENT)AND(CLK50MHZ='1');
          IF (CONT>=1)THEN
          CLK25MHZ<='0';
          CONT<=0;
          ELSE
   CLK25MHZ<='1';
          CONT<=CONT+1;
          END IF;
END PROCESS RELOJ25MHZ ;

RELOJ1MHZ:PROCESS
BEGIN
WAIT UNTIL(CLK50MHZ'EVENT)AND(CLK50MHZ='1');
          IF (CONT2>=25)THEN
          CLK1MHZ<='0';
          IF (CONT2=50)THEN
                    CONT2<=0;
                    ELSE
                    CONT2<=CONT2+1;
                    END IF;
          ELSE
  CLK1MHZ<='1';
          CONT2<=CONT2+1;
          END IF;
END PROCESS RELOJ1MHZ ;

RELOJ100KHZ:PROCESS
BEGIN
WAIT UNTIL(CLK50MHZ'EVENT)AND(CLK50MHZ='1');
          IF (CONT3>=250)THEN
          CLK100KHZ<='0';
          IF (CONT3=500)THEN
                    CONT3<=0;
                    ELSE
                    CONT3<=CONT3+1;
                    END IF;
          ELSE
   CLK100KHZ<='1';
          CONT3<=CONT3+1;
          END IF;
END PROCESS RELOJ100KHZ ;
```

“VGA2.vhd”

```
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.PAQUETE.ALL;

ENTITY VGA2 IS
        PORT
        ( CLK50MHZ,ENA                  : IN STD_LOGIC;
        BLUE2                           : OUT STD_LOGIC_VECTOR( 9 DOWNTO 0);
         REDOUT,GREENOUT,BLUEOUT  : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
          H_SYNC2, V_SYNC2,BLANK2, SYNC2,
          H_SYNC_OUT,V_SYNC_OUT,CLK25MHZ,BLANK,SYNC  : OUT STD_LOGIC;
          OE,CE,LB,UB                   : OUT STD_LOGIC;
          WE                          : OUT STD_LOGIC;
    SRAM_DQ                         : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
          SRAM_ADR                                                  : OUT
STD_LOGIC_VECTOR (17 DOWNTO 0));
END VGA2;

ARCHITECTURE a OF VGA2 IS
        SIGNAL CLKWIRE                  :STD_LOGIC;
        SIGNAL H_SYNC,V_SYNC            :STD_LOGIC;
        SIGNAL VIDEO_ON,VIDEO_ON_V,VIDEO_ON_H :STD_LOGIC;
        SIGNAL H_COUNT,V_COUNT          :STD_LOGIC_VECTOR(9 DOWNTO 0);
        SIGNAL COUNT_ADR                :STD_LOGIC_VECTOR(17 DOWNTO 0);
   SIGNAL BLUES                 :STD_LOGIC_VECTOR(7 DOWNTO 0);

--Se utiliza un componente que divida la frecuencia del reloj de 50mhz a 25mhz
BEGIN
ET :CLKDIVIDE PORT MAP(CLK50MHZ=>CLK50MHZ,CLK25MHZ=>CLKWIRE);
CLK25MHZ<=CLKWIRE;

--La señal VIDEO_ON es equivalente a BLANK del DAC de video ADV7123
VIDEO_ON<=VIDEO_ON_H AND VIDEO_ON_V;
BLANK<=VIDEO_ON;
BLANK2<=VIDEO_ON;

--Para controlar la SRAM se pone en modo de solo escritura--
OE<='0' ;CE<='0'; LB<='0'; UB<='0'; WE<='1';
SYNC<='0'; --SYNC2<='0';
--Se pone lo del contador a las direcciones de la SRAM
SRAM_ADR<=COUNT_ADR;
--Se ponen a unos puesto que solo se utilizara el componente R( 8 bits msb solamente) del RGB
GREENOUT(9 downto 0)<="1111111111";
REDOUT(9 downto 0)<="1111111111";
BLUEOUT(1 DOWNTO 0)<="11";
BLUE2(1 DOWNTO 0)<="11";
--Pone en REDOUT lo que tiene en el puerto si WE=1 osea lectura
BLUES<=SRAM_DQ(7 DOWNTO 0);
```

“VGA2.vhd” Continuación

```
PROCESS
BEGIN
        WAIT UNTIL (CLKWIRE'EVENT) AND (CLKWIRE='1');

IF ENA='1' THEN
        --Generar las señales de tiempos Horizontal y Vertical para la señal de video
        --H_COUNT cuenta pixeles(640+tiempo extra para señales de sincronizacion)
        --
        --HORIZ_SYNC ----------_____------
        --H_COUNT    0       640       659    755   799

        IF (H_COUNT=799) THEN
        H_COUNT<="0000000000";
        ELSE
        H_COUNT<=H_COUNT+1;
        END IF;

        --Generar Señal de sincronizacion Horizontal usando H_COUNT--
        IF (H_COUNT<=755) AND (H_COUNT>=659) THEN
                H_SYNC<='0';
        ELSE
           H_SYNC<='1';
        END IF;
        --V_COUNT cuenta renglones de pixeles (480+ tiempo extra para señales de sincronizacion)

        --VERT_SYNC ----------_______---------
        --V_COUNT   0          480          493-494      524

        IF (V_COUNT>=524) AND (H_COUNT>=799) THEN
           V_COUNT<="0000000000";
        ELSIF (H_COUNT=799) THEN
                V_COUNT<=V_COUNT+1;
        END IF;

        --Generar señal de sincronizacion Vertical usando V_COUNT
        IF (V_COUNT<=494) AND (V_COUNT>=493) THEN
                V_SYNC<='0';
        ELSE
                V_SYNC<='1';
        END IF;

        --Generar Señales de Video en Pantalla para los datos de los pixeles
        IF (H_COUNT<=639) THEN
                VIDEO_ON_H<='1';
        ELSE
                VIDEO_ON_H<='0';
        END IF;


        IF (V_COUNT<=479) THEN
           VIDEO_ON_V<='1';
        ELSE
                VIDEO_ON_V<='0';
        END IF;
```

“VGA2.vhd” Continuación

```
        --Usar contador de 18 bits para direcciones de la SRAM 256kx16 (262,144 localidades de 16
bits)
        --Inicializar contador cada vez que se inicie un nuevo cuadro (posicion 0,0)
        IF (H_COUNT=0) AND (V_COUNT=0) THEN
                COUNT_ADR<="000000000000000000";
                END IF;

        --Usando la parte valida de despliegue (VIDEO_ON='1')
        IF (VIDEO_ON='1')  THEN
                IF (COUNT_ADR=262163) THEN
                COUNT_ADR<="000000000000000000";
                ELSE
                COUNT_ADR<=COUNT_ADR+1;
                END IF;
        END IF;

        V_SYNC_OUT<=V_SYNC;
        H_SYNC_OUT<=H_SYNC;
        V_SYNC2<=V_SYNC;
        H_SYNC2<=H_SYNC;
   BLUEOUT(9 DOWNTO 2)<=BLUES;
   BLUE2(9 DOWNTO 2)<=BLUES;
END IF;


END PROCESS;
END a;
```

“DATINCTRL2.vhd”

```
LIBRARY IEEE;
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY DATINCTRL2 IS
        PORT
(DQ_D         :OUT    STD_LOGIC_VECTOR(15 DOWNTO 0);
DATIN        :IN     STD_LOGIC_VECTOR(7 DOWNTO 0);  --un solo puerto de 8 bits del
Daqpad6507
 OBF,CLK,ENA_D  :      IN            STD_LOGIC;
 ACK              :      OUT    STD_LOGIC;
ADR_D                    :OUT    STD_LOGIC_VECTOR (17 DOWNTO 0);
 WE_D,IDLE               :OUT    STD_LOGIC);
END DATINCTRL2;

ARCHITECTURE a OF DATINCTRL2 IS
   TYPE TIPOESTADO IS (EDOA,EDOB);
        SIGNAL ESTADO:TIPOESTADO;
        SIGNAL CONT1,CONT2:INTEGER:=0 ;
        SIGNAL CONTADR:STD_LOGIC_VECTOR(17 DOWNTO 0);
        SIGNAL ACKWE :STD_LOGIC;

BEGIN

ADR_D<=CONTADR;
DQ_D(15 DOWNTO 8)<="00000000";
--Se escriben los 8 bits de entrada a los LSB de SDRAM_DQ
DQ_D(7 downto 0)<=DATIN WHEN ACKWE='0' ELSE "ZZZZZZZZ";
ACK<=ACKWE;
WE_D<=ACKWE;

HANDSHAKE:PROCESS (CLK,ENA_D)       --HANDSHAKE CON EL DAQPad-6507
BEGIN
IF CLK'EVENT AND CLK='1' THEN
   IF ENA_D='1' THEN
                CASE ESTADO IS
                WHEN EDOA =>
                IF OBF='0' THEN
                IF CONT1=50000 THEN          --Hace el tiempo de OBF=0 a ACK=0
                CONT1<=0;
        ESTADO<=EDOB;
        ELSE
        CONT1<=CONT1+1;
        END IF;
        ELSE
        ESTADO<=EDOA;
                END IF;
```

“DATINCTRL2.vhd” Continuación

```
                WHEN EDOB =>
                IF OBF='0' OR OBF='1' THEN
                        IF CONT2=50000 THEN      --Hace el ancho de ACK
                        CONT2<=0;
                        ESTADO<=EDOA;
                        ELSE
                        CONT2<=CONT2+1;
                        ESTADO<=EDOB;
                        END IF;
                END IF;

                WHEN OTHERS=>
                ESTADO<=EDOA;
        END CASE;
   END IF;
END IF;
END PROCESS HANDSHAKE;

WITH ESTADO SELECT
        ACKWE<='1' WHEN EDOA,
          '0' WHEN EDOB;

--Usar contador de 18 bits para direcciones de la SRAM  (262,144 localidades de 16 bits)
PROCESS(OBF,ENA_D)
BEGIN
IF OBF'EVENT AND OBF='1' THEN
 IF ENA_D='1' THEN
        IF CONTADR=15 THEN
        IDLE<='1';
        ELSE
        IDLE<='0';
        CONTADR<=CONTADR+1;
        END IF;
 END IF;
END IF;
END PROCESS ;
END a;
```

“STARTDATIN.vhd”

```
LIBRARY IEEE,WORK;
USE IEEE. STD_LOGIC_1164.ALL;
USE IEEE. STD_LOGIC_ARITH. ALL;

ENTITY STARTDATIN IS
        PORT
        (CLK            : IN  STD_LOGIC;
         PSH            : IN  STD_LOGIC;
         ENA_D          : OUT STD_LOGIC);
END STARTDATIN;


ARCHITECTURE a OF STARTDATIN IS
        TYPE STATE_TYPE IS (INICIAL,HABILITADO);
        SIGNAL STATE: STATE_TYPE;
BEGIN

        PROCESS (CLK)
        BEGIN
        IF CLK'EVENT AND CLK='1' THEN
                        CASE STATE IS
                           WHEN INICIAL =>
                           IF PSH='0' THEN
                             STATE<=HABILITADO;
                                ELSE
                                  STATE<=INICIAL;
                                END IF;

                                WHEN HABILITADO=>
                                STATE <=HABILITADO;
                        END CASE;
                END IF;
        END PROCESS;

        WITH STATE SELECT
                ENA_D  <='0' WHEN INICIAL,
                                '1' WHEN HABILITADO;


END a;
```

"**BUSES.vhd"**

```
LIBRARY IEEE;
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY BUSES IS
        PORT
        (SRAM_ADR     :OUT STD_LOGIC_VECTOR(17 DOWNTO 0) ;
        ADR_D         :IN STD_LOGIC_VECTOR(17 DOWNTO 0) ;
         ADR_SW       :IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
        DQ_D          :IN STD_LOGIC_VECTOR(15 DOWNTO 0) ;
         SRAM_DQ      :INOUT STD_LOGIC_VECTOR(15 DOWNTO 0);
         DQ_LED       :OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
        IDLE,WE_D     :IN STD_LOGIC;
         WE,WEOUT     :OUT STD_LOGIC;
         CE,OE,UB,LB  :OUT STD_LOGIC);
END BUSES;


ARCHITECTURE a OF BUSES IS

BEGIN
CE<='0'; OE<='0'; UB<='0'; LB<='0';

PROCESS (IDLE,WE_D,ADR_D,DQ_D,SRAM_DQ,ADR_SW)
BEGIN

IF IDLE='0' THEN
        WE<=WE_D;
        WEOUT<=WE_D;  --PARA VERLO EN OSC
        SRAM_ADR<=ADR_D;
        SRAM_DQ<=DQ_D;
        DQ_LED<=SRAM_DQ(3 DOWNTO 0);
ELSE
  SRAM_DQ<="ZZZZZZZZZZZZZZZZ";
  DQ_LED<=SRAM_DQ(3 DOWNTO 0);
  WE<='1';
  WEOUT<='1';
  SRAM_ADR(17 DOWNTO 4)<="00000000000000";
  SRAM_ADR(3 DOWNTO 0)<=ADR_SW;

END IF;
END PROCESS ;

END a;
```

“DATINRAM.vhd”

```
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.PAQUETE.ALL;

ENTITY DATINRAM IS
        PORT
(CLK50MHZ     :IN     STD_LOGIC;
OBF,PSH       :IN     STD_LOGIC;
DATIN         :IN     STD_LOGIC_VECTOR(7 DOWNTO 0);--un solo puerto  de 8 bits aqpad6507
ADR_SW        :IN     STD_LOGIC_VECTOR(3 DOWNTO 0);
DQ_LED,DATIN2                 :OUT    STD_LOGIC_VECTOR(3 DOWNTO 0);
ACK,WE,OE,CE,UB,LB,
 WEOUT,ENA_DOUT,IDLEOUT :OUT    STD_LOGIC;
  SRAM_ADR                    :OUT    STD_LOGIC_VECTOR (17 DOWNTO 0);
  SRAM_DQ                             :INOUT   STD_LOGIC_VECTOR(15 DOWNTO 0));
END DATINRAM;

ARCHITECTURE a OF DATINRAM IS
        SIGNAL IDLES,WE_DS, ENA_DS :STD_LOGIC;
        SIGNAL ADR_DS          :STD_LOGIC_VECTOR (17 DOWNTO 0);
        SIGNAL DQ_DS           :STD_LOGIC_VECTOR (15 DOWNTO 0);

BEGIN

U1:DATINCTRL2
        PORT MAP
        (OBF=>OBF, DATIN=>DATIN, CLK=>CLK50MHZ, ENA_D=>ENA_DS,
         ACK=>ACK, DQ_D=>DQ_DS, ADR_D=>ADR_DS, IDLE=>IDLES, WE_D=>WE_DS);

U2:STARTDATIN
        PORT MAP
   (CLK=>CLK50MHZ, PSH=>PSH, ENA_D=>ENA_DS);

U3:BUSES
        PORT MAP
        (ADR_SW=>ADR_SW, DQ_LED=>DQ_LED, UB=>UB, LB=>LB, CE=>CE, OE=>OE,
         SRAM_ADR=>SRAM_ADR, SRAM_DQ=>SRAM_DQ, WE=>WE, WEOUT=>WEOUT,
         WE_D=>WE_DS, IDLE=>IDLES, ADR_D=>ADR_DS, DQ_D=>DQ_DS);

ENA_DOUT<=ENA_DS;
IDLEOUT<=IDLES;
DATIN2<=DATIN(3 DOWNTO 0);
END a;
```

“matriz-vector.m”

```
clear
clc
%Se descompone la imagen wl_640_409.jpg en sus tres componentes
RGB= imread('C:\Documents and Settings\uabc\Mis documentos\OLAF\Pragramas Simulacion Tarjeta
NI\wl_640_409.jpg');
R=RGB(:,:,1);
G=RGB(:,:,2);
B=RGB(:,:,3);

%Se pasa la Imagen R de mxn a un vector buffer (mnx1) para que se pueda
%usar la funcion DIG_Block_Out
[m,n] = size(R);
l=1;
for  i= 1:m,
    for j = 1:n,
    buffer(l,1)=R(i,j);
     l=l+1;
    end
end

%Se envia el vector buffer haciendo transferecnia handshanking
load ('buffer');
status = DIG_SCAN_Setup (1,1,0,1);
a= DIG_Block_Out(1,1,buffer,260000);
```

“DATINCTRL.vhd”

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY DATINCTRL IS
        PORT
(DQ_D    :OUT   STD_LOGIC_VECTOR(15 DOWNTO 0);
DATIN    :IN    STD_LOGIC_VECTOR(7 DOWNTO 0);    --un solo puerto de 8 bits del Daqpad6507
OBF,CLK25,ENA_D  :IN STD_LOGIC;
ACK                 :OUT   STD_LOGIC;
ADR_D               :OUT   STD_LOGIC_VECTOR (17 DOWNTO 0);
WE_D,IDLE           :OUT   STD_LOGIC);
END DATINCTRL;


ARCHITECTURE a OF DATINCTRL IS
   TYPE TIPOESTADO IS (EDOA,EDOB);
        SIGNAL ESTADO:TIPOESTADO;
        SIGNAL CONT1,CONT2:INTEGER:=0 ;
        SIGNAL CONTADR:STD_LOGIC_VECTOR(17 DOWNTO 0);
        SIGNAL ACKWE :STD_LOGIC;

BEGIN
ADR_D<=CONTADR;
DQ_D(15 DOWNTO 8)<="00000000";
--Se escriben los 8 bits de entrada a los LSB de SDRAM_DQ
DQ_D(7 downto 0)<=DATIN WHEN ACKWE='0' ELSE "ZZZZZZZZ";
ACK<=ACKWE;
WE_D<=ACKWE;

--Proceso para hacer el handshaking necesario entre el Daqpad6507  y el DE2
HANDSHAKE:PROCESS (CLK25,ENA_D)
BEGIN
IF CLK25'EVENT AND CLK25='1' THEN
        IF ENA_D='1' THEN
                CASE ESTADO IS
                WHEN EDOA =>
                IF OBF='0' THEN
                IF CONT1=50000 THEN          --Hace el tiempo de OBF=0 a ACK=0
                CONT1<=0;
        ESTADO<=EDOB;
        ELSE
        CONT1<=CONT1+1;
        END IF;
        ELSE
        ESTADO<=EDOA;
                END IF;
```

“DATINCTRL.vhd” Continuación

```
                WHEN EDOB =>
                IF OBF='0' OR OBF='1' THEN
                        IF CONT2=50000 THEN      --Hace el ancho de ACK
                        CONT2<=0;
                        ESTADO<=EDOA;
                        ELSE
                        CONT2<=CONT2+1;
                        ESTADO<=EDOB;
                        END IF;
                END IF;

                WHEN OTHERS=>
                ESTADO<=EDOA;
        END CASE;
  END IF;
END IF;
END PROCESS HANDSHAKE;

WITH ESTADO SELECT
        ACKWE<='1' WHEN EDOA,
          '0' WHEN EDOB;

--Usar contador de 18 bits para direcciones de la SRAM  (262,144 localidades de 16 bits)
PROCESS(OBF,ENA_D)
BEGIN
IF OBF'EVENT AND OBF='0' THEN
 IF ENA_D='1' THEN
        IF CONTADR=260000 THEN    --para escribir 260000 datos a la SRAM
        IDLE<='1';
        ELSE
        IDLE  <='0';
        CONTADR<=CONTADR+1;
        END IF;
 END IF;
END IF;
END PROCESS ;

END a;
```

“VGA3.vhd”

```
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;


ENTITY VGA3 IS
        PORT
( CLK25,ENA_V                   :IN STD_LOGIC;
REDOUT,GREENOUT,BLUEOUT  :OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
H_SYNC_OUT,V_SYNC_OUT,
BLANK,SYNC,WE_V                 :OUT STD_LOGIC;
 PIXEL_ROW,PIXEL_COLUMN    : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
 DQ_V                           :IN STD_LOGIC_VECTOR(15 DOWNTO 0);
ADR_V                           :OUT STD_LOGIC_VECTOR (17 DOWNTO 0));
END VGA3;

ARCHITECTURE a OF VGA3 IS
        SIGNAL H_SYNC,V_SYNC                            :STD_LOGIC;
        SIGNAL VIDEO_ON,VIDEO_ON_V,VIDEO_ON_H :STD_LOGIC;
        SIGNAL H_COUNT,V_COUNT                          :STD_LOGIC_VECTOR(9 DOWNTO
0);
        SIGNAL COUNT_ADR                                :STD_LOGIC_VECTOR(17 DOWNTO
0);
        SIGNAL REDS                                     :STD_LOGIC_VECTOR(7 DOWNTO
0);
BEGIN

--La señal VIDEO_ON es equivalente a BLANK del DAC de video ADV7123
VIDEO_ON<=VIDEO_ON_H AND VIDEO_ON_V;
BLANK<=VIDEO_ON; WE_V<='1';        SYNC<='0';

--Se pone lo del contador a las direcciones de la SRAM
ADR_V<=COUNT_ADR;
--Se ponen a unos puesto que solo se utilizara el componente R( 8 bits msb solamente) del RGB
GREENOUT<="1111111111";
BLUEOUT<="1111111111";
REDOUT(1 DOWNTO 0)<="11";

--Pone en REDS lo loque tiene en el puerto DQ_V,solo se usaran los 8 MSB de la señal de video
REDOUT(9 A 2)
REDS<=DQ_V(7 DOWNTO 0);

PROCESS
BEGIN
```

"VGA3.vhd" Continuación

```
WAIT UNTIL (CLK25'EVENT) AND (CLK25='1');
--Habilitador para el proceso y poner en Z EL SDRAM_DQ y WE,WE=1 pone a lectura a la SRAM
IF ENA_V='1' THEN

        --Generar las señales de tiempos Horizontal y Vertical para la señal de video
        --H_COUNT cuenta pixeles(640+tiempo extra para señales de sincronizacion)
        --
        --HORIZ_SYNC ----------_____------
        --H_COUNT    0       640       659    755   799

        IF (H_COUNT=799) THEN
        H_COUNT<="0000000000";
        ELSE
        H_COUNT<=H_COUNT+1;
        END IF;

        --Generar Señal de sincronizacion Horizontal usando H_COUNT--
        IF (H_COUNT<=755) AND (H_COUNT>=659) THEN
                H_SYNC<='0';
        ELSE
           H_SYNC<='1';
        END IF;
        --V_COUNT cuenta renglones de pixeles (480+ tiempo extra para señales de sincronizacion)

        --VERT_SYNC ----------_______---------
        --V_COUNT   0          480          493-494      524

        IF (V_COUNT>=524) AND (H_COUNT>=799) THEN
        V_COUNT<="0000000000";
        ELSIF (H_COUNT=799) THEN
                V_COUNT<=V_COUNT+1;
        END IF;

        --Generar señal de sincronizacion Vertical usando V_COUNT
        IF (V_COUNT<=494) AND (V_COUNT>=493) THEN
                V_SYNC<='0';
        ELSE
                V_SYNC<='1';
        END IF;

        --Generar Señales de Video en Pantalla para los datos de los pixeles
        IF (H_COUNT<=639) THEN
                VIDEO_ON_H<='1';
        ELSE
                VIDEO_ON_H<='0';
        END IF;


        IF (V_COUNT<=479) THEN
           VIDEO_ON_V<='1';
        ELSE
                VIDEO_ON_V<='0';
        END IF;
```

“VGA3.vhd” Continuación

```
        --Inicializar el contador de direcciones de la SRAM cuando se regrese a la coordenada 0,0
        IF (H_COUNT=0) AND (V_COUNT=0) THEN
           COUNT_ADR<="000000000000000000";
        END IF;


        --Usar contador de 18 bits para direcciones de la SRAM  (262,144 localidades de 16 bits)
        --Usando la parte valida de despliegue (VIDEO_ON='1')para tomar datos
        IF (VIDEO_ON='1')  THEN
                IF (COUNT_ADR=262163) THEN
                COUNT_ADR<="000000000000000000";
                ELSE
                COUNT_ADR<=COUNT_ADR+1;
                END IF;
        END IF;

        V_SYNC_OUT<=V_SYNC;
   H_SYNC_OUT<=H_SYNC;
        REDOUT(9 DOWNTO 2)<=REDS;


END IF;

END PROCESS;
END a;
```

“CTRL_UNIT.vhd”

```
LIBRARY IEEE;
USE IEEE. STD_LOGIC_1164.ALL;
USE IEEE. STD_LOGIC_ARITH. ALL;

ENTITY CTRL_UNIT IS
        PORT
        (CLK25          : IN  STD_LOGIC;
         IDLE,PSH       : IN  STD_LOGIC;
         ENA_V,ENA_D    : OUT STD_LOGIC);
END CTRL_UNIT;


ARCHITECTURE a OF CTRL_UNIT IS
        TYPE STATE_TYPE IS (INICIAL,ESCRITURA,LECTURA);
        SIGNAL state: STATE_TYPE;
BEGIN


        PROCESS (CLK25)
        BEGIN
        IF CLK25'EVENT AND CLK25='1' THEN
                        CASE state IS
                           WHEN INICIAL =>
                           IF PSH='0' THEN
                             state<=ESCRITURA;
                                END IF;

                                WHEN ESCRITURA =>
                                        IF IDLE='1' THEN
                                                state <=LECTURA;
                                        END IF;

                                WHEN LECTURA=>
                                state <=LECTURA;

                        END CASE;
                END IF;
        END PROCESS;

        WITH state SELECT
                ENA_D <='0' WHEN INICIAL,
                                '1' WHEN ESCRITURA,
                                '0' WHEN LECTURA;

        WITH state SELECT
                ENA_V <='0' WHEN INICIAL,
                                '0' WHEN ESCRITURA,
                                '1' WHEN LECTURA;

END a;
```

“BUSCTRL.vhd”

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY BUSCTRL IS
        PORT
(SRAM_ADR           :OUT STD_LOGIC_VECTOR(17 DOWNTO 0) ;
ADR_D,ADR_V         :IN STD_LOGIC_VECTOR(17 DOWNTO 0) ;
DQ_D                :IN STD_LOGIC_VECTOR(15 DOWNTO 0) ;
SRAM_DQ             :INOUT STD_LOGIC_VECTOR(15 DOWNTO 0);
DQ_V                :OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
IDLE,WE_D,WE_V      :IN STD_LOGIC;
WE                  :OUT STD_LOGIC;
CE,OE,UB,LB         :OUT STD_LOGIC);
END BUSCTRL;


ARCHITECTURE a OF BUSCTRL IS

BEGIN
CE<='0'; OE<='0'; UB<='0'; LB<='0';

PROCESS (IDLE,WE_D,WE_V,ADR_D,ADR_V,DQ_D,SRAM_DQ)
BEGIN

IF IDLE='0' THEN
        WE<=WE_D;
        SRAM_ADR<=ADR_D;
        SRAM_DQ<=DQ_D;
ELSE
  WE<=WE_V;
  SRAM_ADR<=ADR_V;
  SRAM_DQ<="ZZZZZZZZZZZZZZZZ";
  DQ_V<=SRAM_DQ;
END IF;
END PROCESS ;

END a;
```

"DESP_VGA.vhd"

```
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.PAQUETE.ALL;

ENTITY DESP_VGA IS
        PORT
(CLK50                  :IN     STD_LOGIC;
 OBF,PSH                :IN     STD_LOGIC;
DATIN                   :IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
CLKVGA,ACK,
WE,OE,CE,UB,LB          :OUT    STD_LOGIC;
SRAM_ADR                :OUT    STD_LOGIC_VECTOR (17 DOWNTO 0);
SRAM_DQ                 :INOUT  STD_LOGIC_VECTOR(15 DOWNTO 0);
REDOUT,GREENOUT,BLUEOUT     :OUT    STD_LOGIC_VECTOR(9 DOWNTO 0);
H_SYNC_OUT,V_SYNC_OUT,
BLANK,SYNC                      :OUT      STD_LOGIC);
END DESP_VGA;

ARCHITECTURE a OF DESP_VGA IS
        SIGNAL CLK25S,WE_DS,WE_VS,IDLES,ENA_VS,ENA_DS :STD_LOGIC;
        SIGNAL ADR_DS,ADR_VS : STD_LOGIC_VECTOR (17 DOWNTO 0);
        SIGNAL DQ_VS,DQ_DS  : STD_LOGIC_VECTOR (15 DOWNTO 0);

BEGIN
CLKVGA<=CLK25S;

U1:CLKDIVIDE
        PORT MAP
        (CLK50MHZ=>CLK50, CLK25MHZ=>CLK25S);

U2:DATINCTRL
        PORT MAP
(OBF=>OBF, ACK=>ACK, DATIN=>DATIN, CLK25=>CLK25S, ENA_D=>ENA_DS,
IDLE=>IDLES,
   DQ_D=>DQ_DS, ADR_D=>ADR_DS, WE_D=>WE_DS);

U3:CTRL_UNIT
  PORT MAP
        (CLK25=>CLK25S, IDLE=>IDLES, PSH=>PSH, ENA_V=>ENA_VS, ENA_D=>ENA_DS);

U4:VGA3
        PORT MAP
   (CLK25=>CLK25S, ENA_V=>ENA_VS, REDOUT=>REDOUT, GREENOUT=>GREENOUT,
    BLUEOUT=>BLUEOUT, H_SYNC_OUT=>H_SYNC_OUT,V_SYNC_OUT=>V_SYNC_OUT,
    BLANK=>BLANK, SYNC=>SYNC, WE_V=>WE_VS, DQ_V=>DQ_VS, ADR_V=>ADR_VS);

U5:BUSCTRL
        PORT MAP
        (SRAM_ADR=>SRAM_ADR, SRAM_DQ=>SRAM_DQ, ADR_D=>ADR_DS,
ADR_V=>ADR_VS, DQ_V=>DQ_VS, DQ_D=>DQ_DS,
         IDLE=>IDLES, WE_D=>WE_DS, WE_V=>WE_VS, WE=>WE, CE=>CE, OE=>OE,
UB=>UB, LB=>LB);
END a;
```

Apéndice B Códigos en VHDL realizados de la sección 3.2.1 a la 3.2.6

"I2C_WR2.vhd"

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
ENTITY I2C_WR2 IS
        PORT
        (       CLK,ENA         : IN    STD_LOGIC;
                FIN                     : OUT STD_LOGIC;
                SDA,SCL         : INOUT     STD_LOGIC);
END I2C_WR2;

ARCHITECTURE a OF I2C_WR2 IS
CONSTANT WRITE_ADR  :STD_LOGIC_VECTOR (7 DOWNTO 0):="10111010";--Slave Device 8-
bit adr. con el LSB=0 (Write)
CONSTANT REG_ADR    :STD_LOGIC_VECTOR (7 DOWNTO 0):="00001001"; --REGISTRO
CONSTANT DATA_MSB   :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000010";
CONSTANT DATA_LSB   :STD_LOGIC_VECTOR (7 DOWNTO 0):="10000100";

TYPE STATE_TYPE IS (IDLE,START_A,START_B,START_C,START_D,WR_A,WR_B,WR_C,
WR_D, ACK_A,ACK_B,ACK_C,ACK_D,STOP_A,STOP_B,STOP_C);
        SIGNAL STATE: STATE_TYPE;
        SIGNAL CNT :INTEGER RANGE 0 TO 7 ;
        SIGNAL CNT2: INTEGER RANGE 0 TO 3:=0;
        SIGNAL SCLo,SDAo :STD_LOGIC;
        SIGNAL DATO : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
        PROCESS (CLK)
        BEGIN
IF clk'EVENT AND clk = '1' THEN
                CASE STATE IS
                        WHEN  IDLE=>
                          CNT<=7;
                                CASE CNT2 IS
                                        WHEN 0 =>
                                                DATO<=WRITE_ADR;
                                        WHEN 1 =>
                                                DATO<=REG_ADR;
                                        WHEN 2 =>
                                                DATO<=DATA_MSB;
                                        WHEN 3 =>
                                                DATO<=DATA_LSB;
                                END CASE;
                        IF ENA='1' THEN
                                IF CNT2=0 THEN
                                STATE <=START_A;
                                ELSE
                                STATE<=WR_A;
                                END IF;
                        ELSE
                                STATE<=IDLE;
                        END IF;
```

"I2C_WR2.vhd" Continuación

```
                        when start_a =>
                                STATE<=START_B;
                        when start_b =>
                                STATE<=START_C;
                        when start_c =>
                                STATE<=START_D;
                        when start_d =>
                                STATE<=WR_A;
                        WHEN WR_A =>
        STATE<=WR_B;
                        WHEN WR_B =>
        STATE<=WR_C;
                        WHEN WR_C =>
        STATE<=WR_D;
                        WHEN WR_D =>

                                IF (CNT=0) THEN
                                        STATE <=ACK_A;
                                        ELSE
                CNT<=CNT-1;
                                   STATE<=WR_A;
                                        END IF;
                 when ACK_A =>
                                STATE<=ACK_B;
                 when ACK_B =>
                                STATE<=ACK_C;
                 when ACK_C =>
                   IF SDA='0' THEN
                                STATE<=ACK_D;
                                ELSE
                                STATE<=ACK_C;
                                END IF;
                 when ACK_D =>
                                IF CNT2=3 THEN
                                  CNT2<=0;
                                  STATE<=STOP_A;
                                ELSE
                                        CNT2<=CNT2+1;
                                   STATE<=IDLE;
                                END IF;

                when stop_a =>
                                state <= stop_b;
                        when stop_b =>
                                state <= stop_c;
                        when stop_c =>
                                state <=IDLE ;
                        END CASE;
END IF;
        END PROCESS;
```

“I2C_WR2.vhd” Continuación

```
WITH state SELECT
                SCL<='0'   WHEN IDLE,
          '1'  WHEN START_A,
          '1'  WHEN START_B,
          '1'  WHEN START_C,
          '0'  WHEN START_D,
          '0'  WHEN WR_A,
          '1'  WHEN WR_B,
          '1'  WHEN WR_C,
          '0'  WHEN WR_D,
          '0'  WHEN ACK_A,
          '1'  WHEN ACK_B,
          '1'  WHEN ACK_C,
          '0'  WHEN ACK_D,
          '0'  WHEN STOP_A,
          '1'  WHEN STOP_B,
          '1'  WHEN STOP_C;
WITH state SELECT
                  SDA<='0'  WHEN IDLE,
                            '1'  WHEN START_A,
                           '1'   WHEN START_B,
                           '0'   WHEN START_C,
                           '0'   WHEN START_D,
           DATO(CNT) WHEN WR_A,
           DATO(CNT) WHEN WR_B,
           DATO(CNT) WHEN WR_C,
           DATO(CNT) WHEN WR_D,
          'Z'   WHEN ACK_A,
          'Z'   WHEN ACK_B,
          'Z'   WHEN ACK_C,
          'Z'   WHEN ACK_D,
          '0'   WHEN STOP_A,
          '0'   WHEN STOP_B,
          '1'   WHEN STOP_C;
WITH state SELECT
                FIN<= '0'   WHEN IDLE,
                           '0'       WHEN START_A,
                           '0'       WHEN START_B,
                           '0'       WHEN START_C,
                           '0'       WHEN START_D,
                           '0'   WHEN WR_A,
          '0'   WHEN WR_B,
          '0'   WHEN WR_C,
          '0'   WHEN WR_D,
          '0'   WHEN ACK_A,
          '0'   WHEN ACK_B,
          '0'   WHEN ACK_C,
          '0'   WHEN ACK_D,
          '1'   WHEN STOP_A,
          '1'   WHEN STOP_B,
          '1'   WHEN STOP_C;
END a;
```

“I2C_WR2A.vhd”

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
ENTITY I2C_WR2A IS
        PORT
        (CLK,ENA                          : IN          STD_LOGIC;
        FIN,SLV_ENA,SCLO2                 : OUT  STD_LOGIC;
        SDA,SCL                           : INOUT STD_LOGIC;
        REG_ADR,DATA_MSB,DATA_LSB : IN STD_LOGIC_VECTOR(7 DOWNTO 0));
END I2C_WR2A;
ARCHITECTURE a OF I2C_WR2A IS
CONSTANT WRITE_ADR  :STD_LOGIC_VECTOR (7 DOWNTO 0):="10111010";--Slave Device 8
bit,  --adr. con el LSB=0 (Write)

TYPE STATE_TYPE IS (IDLE,START_A,START_B,START_C,START_D,WR_A,WR_B,WR_C,
WR_D, ACK_A,ACK_B,ACK_C,ACK_D,STOP_A,STOP_B,STOP_C);
        SIGNAL STATE: STATE_TYPE;
        SIGNAL CNT :INTEGER RANGE 0 TO 7 ;
        SIGNAL CNT2: INTEGER RANGE 0 TO 3:=0;
        SIGNAL SCLo,SDAo :STD_LOGIC;
        SIGNAL DATO : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
        PROCESS (CLK)
        BEGIN
IF clk'EVENT AND clk = '1' THEN
        IF ENA='1' THEN
                CASE STATE IS
                        WHEN  IDLE=>
                          CNT<=7;
                                        CASE CNT2 IS
                                                WHEN 0 =>
                                                        DATO<=WRITE_ADR;
                                                WHEN 1 =>
                                                        DATO<=REG_ADR;
                                                WHEN 2 =>
                                                        DATO<=DATA_MSB;
                                                WHEN 3 =>
                                                        DATO<=DATA_LSB;
                                        END CASE;
                                        IF CNT2=0 THEN
                                        STATE <=START_A;
                                        ELSE
                                        STATE<=WR_A;
                                        END IF;
```

“I2C_WR2A.vhd” Continuación

```
                        when start_a =>
                                STATE<=START_B;
                        when start_b =>
                                STATE<=START_C;
                        when start_c =>
                                STATE<=START_D;
                        when start_d =>
                                STATE<=WR_A;
                        WHEN WR_A =>
        STATE<=WR_B;
                        WHEN WR_B =>
        STATE<=WR_C;
                        WHEN WR_C =>
        STATE<=WR_D;
                        WHEN WR_D =>
                                IF (CNT=0) THEN
                                        STATE <=ACK_A;
                                        ELSE
                CNT<=CNT-1;
                                   STATE<=WR_A;
                                        END IF;
                 when ACK_A =>
                                STATE<=ACK_B;
                 when ACK_B =>
                                STATE<=ACK_C;
--aqui
                 when ACK_C =>
                   IF SDA='0' THEN
                                STATE<=ACK_D;
                                ELSE
                                STATE<=ACK_C;
                                END IF;
                when ACK_D =>
                                IF CNT2=3 THEN
                                  CNT2<=0;
                                  STATE<=STOP_A;
                                ELSE
                                        CNT2<=CNT2+1;
                                   STATE<=IDLE;
                                END IF;
                when stop_a =>
                                state <= stop_b;
                        when stop_b =>
                                state <= stop_c;
                        when stop_c =>
                                state <=IDLE ;
                        END CASE;
        END IF;
END IF;
        END PROCESS;
```

“I2C_WR2A.vhd” Continuación

```
WITH state SELECT
                SCLo<='0'   WHEN IDLE,
                            '1'       WHEN START_A,
                            '1'       WHEN START_B,
                            '1'       WHEN START_C,
                            '0'       WHEN START_D,
                            '0'   WHEN WR_A,
        '1'   WHEN WR_B,
        '1'   WHEN WR_C,
        '0'   WHEN WR_D,
         '0'   WHEN ACK_A,
        '1'   WHEN ACK_B,
        '1'   WHEN ACK_C,
        '0'   WHEN ACK_D,
        '0'   WHEN STOP_A,
        '1'   WHEN STOP_B,
        '1'   WHEN STOP_C;


WITH state SELECT
                SDAo<='0'   WHEN IDLE,
                    '1'    WHEN START_A,
                            '1'   WHEN START_B,
                            '0'       WHEN START_C,
                            '0'       WHEN START_D,

         DATO(CNT) WHEN WR_A,
         DATO(CNT) WHEN WR_B,
         DATO(CNT) WHEN WR_C,
         DATO(CNT) WHEN WR_D,
         'Z'   WHEN ACK_A,
        'Z'   WHEN ACK_B,
        'Z'   WHEN ACK_C,
        'Z'   WHEN ACK_D,
        '0'   WHEN STOP_A,
        '0'   WHEN STOP_B,
        '1'   WHEN STOP_C;
WITH state SELECT
                FIN<= '0'   WHEN IDLE,
        '0'   WHEN START_A,
        '0'   WHEN START_B,
        '0'   WHEN START_C,
        '0'   WHEN START_D,
        '0'   WHEN WR_A,
        '0'   WHEN WR_B,
        '0'   WHEN WR_C,
        '0'   WHEN WR_D,
        '0'   WHEN ACK_A,
        '0'   WHEN ACK_B,
        '0'   WHEN ACK_C,
        '0'   WHEN ACK_D,
        '1'   WHEN STOP_A,
        '1'   WHEN STOP_B,
        '0'   WHEN STOP_C;
```

“I2C_WR2A.vhd” Continuación

```
WITH state SELECT
        SLV_ENA<= '0'   WHEN IDLE,
        '0'         WHEN START_A,
        '0'         WHEN START_B,
        '0'         WHEN START_C,
        '0'         WHEN START_D,
        '1'   WHEN WR_A,
        '1'   WHEN WR_B,
        '1'   WHEN WR_C,
        '1'   WHEN WR_D,
        '1'   WHEN ACK_A,
        '1'   WHEN ACK_B,
        '1'   WHEN ACK_C,
        '1'   WHEN ACK_D,
        '0'   WHEN STOP_A,
        '0'   WHEN STOP_B,
        '0'   WHEN STOP_C;
SDA<= '0' WHEN (SDAo = '0') ELSE 'Z';
SCL<= '0' WHEN (SCLo = '0') ELSE 'Z';
SCLO2<=SCLo;
END a;
```

"I2C_CNTR.vhd"

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
ENTITY I2C_CNTR IS
        PORT
        (CLK, PSH, FIN : IN     STD_LOGIC;
        ENA             : OUT   STD_LOGIC;
    REG_ADR,DATA_MSB,DATA_LSB: OUT STD_LOGIC_VECTOR(7 DOWNTO 0) );
END I2C_CNTR;
ARCHITECTURE a OF I2C_CNTR IS
--Tabla de constantes o valores que se deben de escribir a los registros del MT9T001--
CONSTANT REG_ADR0   :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000001"; --Reg0x01
CONSTANT DATA_MSB0  :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000001"; --DATA 0x0164 Indica inicio de row
CONSTANT DATA_LSB0  :STD_LOGIC_VECTOR (7 DOWNTO 0):="01100100"; --en 356 (debe ser par)
CONSTANT REG_ADR1   :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000010"; --Reg0x02
CONSTANT DATA_MSB1  :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000010"; --DATA 0x0238 Indica inicio de col
CONSTANT DATA_LSB1  :STD_LOGIC_VECTOR (7 DOWNTO 0):="00111000"; --en 568 (debe ser par)
CONSTANT REG_ADR2   :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000011";  --Reg0x03=479        default(0x05FF)
CONSTANT DATA_MSB2  :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000001";  --DATA 0x01DF Indica Row size
CONSTANT DATA_LSB2  :STD_LOGIC_VECTOR (7 DOWNTO 0):="11011111";  --en 479= 480-1 (debe ser impar)
CONSTANT REG_ADR3   :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000100";  --Reg0x04=639        default(0x07FF)
CONSTANT DATA_MSB3  :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000010";  --DATA 0x027F Indica Row size
CONSTANT DATA_LSB3  :STD_LOGIC_VECTOR (7 DOWNTO 0):="01111111";  --en 639= 640-1 (debe ser impar)
CONSTANT REG_ADR4   :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000101";  --Reg0x05=526 ;default(0x008E)
CONSTANT DATA_MSB4  :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000010";  --DATA 0x024D Indica 589 pxlcks de;
CONSTANT DATA_LSB4  :STD_LOGIC_VECTOR (7 DOWNTO 0):="01001101";  --Horizontal Blanking
CONSTANT REG_ADR5   :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000110";  --Reg0x06=44         default(0x0019)
CONSTANT DATA_MSB5  :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000000";  --DATA 0x002C Indica 44 tROWs de
CONSTANT DATA_LSB5  :STD_LOGIC_VECTOR (7 DOWNTO 0):="00101100";  --Vertical Blanking
CONSTANT REG_ADR6   :STD_LOGIC_VECTOR (7 DOWNTO 0):="00001001";  --Reg0x09          default(0X0619)
CONSTANT DATA_MSB6  :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000001";  --DATA 0x01E0=480; DATA
0x01E0=480
CONSTANT DATA_LSB6  :STD_LOGIC_VECTOR (7 DOWNTO 0):="11100000";                 --Debe ser <496
CONSTANT REG_ADR7   :STD_LOGIC_VECTOR (7 DOWNTO 0):="00101011";  --Reg0x2B          default(0X0008)
CONSTANT DATA_MSB7  :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000001";  --DATA 0x000A
CONSTANT DATA_LSB7  :STD_LOGIC_VECTOR (7 DOWNTO 0):="01100000";
CONSTANT REG_ADR8   :STD_LOGIC_VECTOR (7 DOWNTO 0):="00101100";  --Reg0x2C          default(0X0008)
CONSTANT DATA_MSB8  :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000001";  --DATA 0x0018
CONSTANT DATA_LSB8  :STD_LOGIC_VECTOR (7 DOWNTO 0):="01100000";
CONSTANT REG_ADR9   :STD_LOGIC_VECTOR (7 DOWNTO 0):="00101101";  --Reg0x2D          default(0X0008)
CONSTANT DATA_MSB9  :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000001";  --DATA 0x000A
CONSTANT DATA_LSB9  :STD_LOGIC_VECTOR (7 DOWNTO 0):="01100000";
CONSTANT REG_ADR10  :STD_LOGIC_VECTOR (7 DOWNTO 0):="00101110";  --Reg0x2E          default(0X0008)
CONSTANT DATA_MSB10 :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000001";  --DATA 0x000A
CONSTANT DATA_LSB10 :STD_LOGIC_VECTOR (7 DOWNTO 0):="01100000";
CONSTANT REG_ADR11  :STD_LOGIC_VECTOR (7 DOWNTO 0):="00110101";  --Reg0x35          default(0X0008)
CONSTANT DATA_MSB11 :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000001";  --DATA 0x000A
CONSTANT DATA_LSB11 :STD_LOGIC_VECTOR (7 DOWNTO 0):="01100000";
CONSTANT REG_ADR12  :STD_LOGIC_VECTOR (7 DOWNTO 0):="01100000";  --Reg0x60          default(0X0020)
CONSTANT DATA_MSB12 :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000000";  --DATA 0x0008
CONSTANT DATA_LSB12 :STD_LOGIC_VECTOR (7 DOWNTO 0):="00100000";
CONSTANT REG_ADR13  :STD_LOGIC_VECTOR (7 DOWNTO 0):="01100001";  --Reg0x61          default(0X0020)
CONSTANT DATA_MSB13 :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000000";  --DATA 0x0008
CONSTANT DATA_LSB13 :STD_LOGIC_VECTOR (7 DOWNTO 0):="00100000";
CONSTANT REG_ADR14  :STD_LOGIC_VECTOR (7 DOWNTO 0):="01100011";  --Reg0x63          default(0X0020)
CONSTANT DATA_MSB14 :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000000";  --DATA 0x0015
```

"I2C_CNTR.vhd" Continuación

```
CONSTANT DATA_LSB14  :STD_LOGIC_VECTOR (7 DOWNTO 0):="00100000";
CONSTANT REG_ADR15   :STD_LOGIC_VECTOR (7 DOWNTO 0):="01100100";  --Reg0x64          default(0X0020)
CONSTANT DATA_MSB15  :STD_LOGIC_VECTOR (7 DOWNTO 0):="00000000";  --DATA 0x001C
CONSTANT DATA_LSB15  :STD_LOGIC_VECTOR (7 DOWNTO 0):="00100000";

        TYPE STATE_TYPE IS (A,AA,B,C,IDLE);
        SIGNAL STATE: STATE_TYPE;
        SIGNAL CNT : INTEGER RANGE 0 TO 15 :=0;
BEGIN
        PROCESS (CLK)
        BEGIN
                IF CLK'EVENT AND CLK = '1' THEN
                        CASE STATE IS
                                WHEN A =>
                                        IF PSH='0' THEN
                                                STATE <=AA;
                                        ELSE
                                           STATE<=A;
                                        END IF;
                                WHEN AA =>
                                        IF PSH='1' THEN
                                                STATE <=B;
                                        ELSE
                                           STATE<=AA;
                                        END IF;
                                WHEN B =>
                                        CASE CNT IS
                                                WHEN 0 =>
                                                        REG_ADR<=REG_ADR0;
                                                        DATA_MSB<=DATA_MSB0;
                                                        DATA_LSB<=DATA_LSB0;
                                                WHEN 1 =>
                                                        REG_ADR<=REG_ADR1;
                                                        DATA_MSB<=DATA_MSB1;
                                                        DATA_LSB<=DATA_LSB1;
                                                WHEN 2 =>
                                                        REG_ADR<=REG_ADR2;
                                                        DATA_MSB<=DATA_MSB2;
                                                        DATA_LSB<=DATA_LSB2;
                                                WHEN 3 =>
                                                        REG_ADR<=REG_ADR3;
                                                        DATA_MSB<=DATA_MSB3;
                                                        DATA_LSB<=DATA_LSB3;
                                                WHEN 4 =>
                                                        REG_ADR<=REG_ADR4;
                                                        DATA_MSB<=DATA_MSB4;
                                                        DATA_LSB<=DATA_LSB4;
                                        WHEN 5 =>
                                                        REG_ADR<=REG_ADR5;
                                                        DATA_MSB<=DATA_MSB5;
                                                        DATA_LSB<=DATA_LSB5;
                                        WHEN 6 =>
                                                        REG_ADR<=REG_ADR6;
                                                        DATA_MSB<=DATA_MSB6;
                                                        DATA_LSB<=DATA_LSB6;
```

“I2C_CNTR.vhd” Continuación

```
                                        WHEN 7 =>
                                                REG_ADR<=REG_ADR7;
                                                DATA_MSB<=DATA_MSB7;
                                                DATA_LSB<=DATA_LSB7;
                                        WHEN 8 =>
                                                REG_ADR<=REG_ADR8;
                                                DATA_MSB<=DATA_MSB8;
                                                DATA_LSB<=DATA_LSB8;
                                        WHEN 9 =>
                                                REG_ADR<=REG_ADR9;
                                                DATA_MSB<=DATA_MSB9;
                                                DATA_LSB<=DATA_LSB9;
                                        WHEN 10 =>
                                                REG_ADR<=REG_ADR10;
                                                DATA_MSB<=DATA_MSB10;
                                                DATA_LSB<=DATA_LSB10;
                                        WHEN 11 =>
                                                REG_ADR<=REG_ADR11;
                                                DATA_MSB<=DATA_MSB11;
                                                DATA_LSB<=DATA_LSB11;
                                        WHEN 12=>
                                                REG_ADR<=REG_ADR12;
                                                DATA_MSB<=DATA_MSB12;
                                                DATA_LSB<=DATA_LSB12;
                                        WHEN 13 =>
                                                REG_ADR<=REG_ADR13;
                                                DATA_MSB<=DATA_MSB13;
                                                DATA_LSB<=DATA_LSB13;
                                        WHEN 14 =>
                                                REG_ADR<=REG_ADR14;
                                                DATA_MSB<=DATA_MSB14;
                                                DATA_LSB<=DATA_LSB14;
                                        WHEN 15 =>
                                                REG_ADR<=REG_ADR15;
                                                DATA_MSB<=DATA_MSB15;
                                                DATA_LSB<=DATA_LSB15;
                                END CASE;
                                STATE<=C;
                        WHEN C =>
                                IF FIN='1' THEN
                                        STATE<=IDLE;
                                ELSE
                                        STATE<=C;
                                END IF;
                        WHEN IDLE =>
                                IF FIN='0' THEN
                                        IF CNT=15 THEN
                                                CNT<=0;
                                                STATE<=A;
                                        ELSE
                                                CNT<=CNT+1;
                                        STATE<=B;
                                                END IF;
                                END IF;
        END CASE;
```

“I2C_CNTR.vhd” Continuación

```
                END IF;
        END PROCESS;

        WITH STATE SELECT
                ENA   <= '0'    WHEN A,
                         '0'    WHEN AA,
                         '1'    WHEN C,
                         '1'    WHEN B,
                         '1'    WHEN IDLE;
END a;
```

“I2C_SLAVE.vhd”

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY I2C_SLAVE IS
        PORT
        (SLV_ENA,SCLO2      :IN    STD_LOGIC;
         SDA          :OUT      STD_LOGIC);
END I2C_SLAVE;

ARCHITECTURE a OF I2C_SLAVE IS
        SIGNAL CNT,CNT2:INTEGER RANGE 0 TO 8:=0;
        SIGNAL START,SDAA,SDAB,SDAO: STD_LOGIC;
BEGIN
SDAO<=SDAA OR SDAB;
SDA<= '0' WHEN (SDAO = '0') ELSE 'Z';
--SDA<=SDAO;

PROCESS(SCLO2,SLV_ENA)
BEGIN
IF SLV_ENA='1' THEN
  IF SCLO2'EVENT AND SCLO2='1' THEN

                        IF CNT=8 THEN
                                SDAA<='0';
                                CNT<=0;
                        ELSE
                        SDAA<='1';
                        CNT<=CNT+1;
                    END IF;
        END IF;
ELSE
        SDAA<='1';
END IF;
END PROCESS;

PROCESS(SCLO2,SLV_ENA)
BEGIN
IF SLV_ENA='1' THEN
        IF SCLO2'EVENT AND SCLO2='0' THEN
                        IF CNT2=8 THEN
                                SDAB<='1';
                                CNT2<=0;
                        ELSE
                        SDAB<='0';
                        CNT2<=CNT2+1;
                    END IF;
        END IF;
ELSE
SDAB<='1';
END IF;
END PROCESS;
END a;
```

“I2C_ESCRITURA.vhd”

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.PAQUETE.ALL;

ENTITY I2C_ESCRITURA IS
        PORT
        (CLK50MHZ,PSH           : IN      STD_LOGIC;
--      ENAO,FINO,SLV_ENAO      : OUT     STD_LOGIC;
        NADA                    : OUT     STD_LOGIC_VECTOR(12 DOWNTO 0);
--      SDASLV,
        SDA,SCL                 : INOUT STD_LOGIC);
        END I2C_ESCRITURA;

ARCHITECTURE a OF I2C_ESCRITURA IS
SIGNAL  REG_ADRS,DATA_MSBS,DATA_LSBS :STD_LOGIC_VECTOR(7 DOWNTO 0);
--SIGNAL  SDAS,SLV_ENAS,SCLO2S,
SIGNAL ENAS,FINS,CLKWIRE: STD_LOGIC;
BEGIN
--FINO<=FINS;
--ENAO<=ENAS;
--SLV_ENAO<=SLV_ENAS;
--SDA<=SDAS;
NADA<="ZZZZZZZZZZZZZ";

U1:I2C_CNTR
        PORT MAP
        (PSH=>PSH,CLK=>CLKWIRE,FIN=>FINS,REG_ADR=>REG_ADRS,DATA_MSB=>DATA
_MSBS,DATA_LSB=>DATA_LSBS,ENA=>ENAS);

U2:I2C_WR2A
        PORT MAP
(CLK=>CLKWIRE,REG_ADR=>REG_ADRS,DATA_MSB=>DATA_MSBS,DATA_LSB=>DATA_L
SBS,FIN=>FINS,ENA=>ENAS,
          SDA=>SDA,SCL=>SCL
        --SLV_ENA=>SLV_ENAS,SCLO2=>SCLO2S);

--U3:I2C_SLAVE
--      PORT MAP
--   (SDA=>SDASLV,SCLO2=>SCLO2S,SLV_ENA=>SLV_ENAS);

U4:CLKDIVIDE
        PORT MAP
        (CLK50MHZ=>CLK50MHZ,CLK100KHZ=>CLKWIRE);
END a;
```

“DINCMOS.vhd”

```
LIBRARY IEEE;
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY DINCMOS IS
        PORT(
DQ_D                    :INOUT   STD_LOGIC_VECTOR(15 DOWNTO 0);
DATIN                   :IN STD_LOGIC_VECTOR(9 DOWNTO 0);
PXCLK,ENA_D,
LV,FV,CLK25             :IN         STD_LOGIC;
ADR_D                   :OUT    STD_LOGIC_VECTOR (17 DOWNTO 0);
 WE_D,IDLE,FVSIGO       :OUT    STD_LOGIC);
END DINCMOS;

ARCHITECTURE a OF DINCMOS IS
   TYPE TIPOESTADO IS (EDOA,EDOB,EDOC);
        SIGNAL ESTADO:TIPOESTADO;
        SIGNAL CONTADR:STD_LOGIC_VECTOR(17 DOWNTO 0);
        SIGNAL WES,FVSIG:STD_LOGIC;
BEGIN
WE_D<=WES;
ADR_D<=CONTADR;
DQ_D(15 DOWNTO 10)<="000000";
DQ_D(9 downto 0)<=DATIN WHEN WES='0' ELSE "ZZZZZZZZZZ";
FVSIGO<=FVSIG;

FRAMEBEGIN:PROCESS(CLK25,ENA_D)
BEGIN
        IF CLK25'EVENT AND CLK25='1' THEN
                IF ENA_D='1' THEN
                        CASE ESTADO IS
                                WHEN EDOA =>
                           IF FV='0' THEN
                                        ESTADO<=EDOB;
                                ELSE
                                        ESTADO<=EDOA;
                                END IF;

                                WHEN EDOB =>
                             IF FV='1' THEN
                                ESTADO<=EDOC;
                                ELSE
                                ESTADO<=EDOB;
                             END IF;
```

“DINCMOS.vhd” Continuación

```
                                        WHEN EDOC =>
                                    IF FV='0' THEN
                                        ESTADO<=EDOA;
                                        ELSE
                                        ESTADO<=EDOC;
                                    END IF;

                                END CASE;
                        END IF;
                END IF;
END PROCESS FRAMEBEGIN;

WITH ESTADO SELECT
        FVSIG<='0' WHEN EDOA,
                '0' WHEN EDOB,
                '1' WHEN EDOC;


--Usar contador de 18 bits para direcciones de la SRAM  (262,144 localidades de 16 bits)
PROCESS(PXCLK)
BEGIN
IF PXCLK'EVENT AND PXCLK='0' THEN
        IF FVSIG='1' AND LV='1' THEN
                IF CONTADR=20 THEN
                IDLE<='1';
                ELSE
                IDLE  <='0';
                CONTADR<=CONTADR+1;
                END IF;
        END IF;
END IF;
        END PROCESS ;

PROCESS(PXCLK)
BEGIN
        IF FVSIG='1' AND LV='1' THEN
                WES<=NOT PXCLK;
        ELSE
                WES<='1';
        END IF;
END PROCESS;
END a;
```

"DESP_VGA2.vhd"

```
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.PAQUETE.ALL;
ENTITY DESP_VGA2 IS
        PORT
(CLK50,PXCLK  :IN     STD_LOGIC;
PSH,LV,FV       :IN     STD_LOGIC;
DATIN           :IN     STD_LOGIC_VECTOR(9 DOWNTO 0);--Bus de datos del Sensor MT9T001
CLKVGA,WE,
 OE,CE,UB,LB                                    :OUT    STD_LOGIC;
SRAM_ADR                                        :OUT    STD_LOGIC_VECTOR (17 DOWNTO 0);
SRAM_DQ                                         :INOUT  STD_LOGIC_VECTOR(15 DOWNTO
0);
REDOUT,GREENOUT,BLUEOUT         :OUT    STD_LOGIC_VECTOR(9 DOWNTO 0);
H_SYNC_OUT,V_SYNC_OUT,
BLANK,SYNC                      :OUT     STD_LOGIC);
END DESP_VGA2;

ARCHITECTURE a OF DESP_VGA2 IS
        SIGNAL CLK25S,WE_DS,WE_VS,IDLES,ENA_VS,ENA_DS :STD_LOGIC;
        SIGNAL ADR_DS,ADR_VS : STD_LOGIC_VECTOR (17 DOWNTO 0);
        SIGNAL DQ_VS,DQ_DS  : STD_LOGIC_VECTOR (15 DOWNTO 0);
BEGIN

U1:CLKDIVIDE
        PORT MAP
        (CLK50MHZ=>CLK50, CLK25MHZ=>CLK25S);

U2:DINCMOS
        PORT MAP
   (PXCLK=>PXCLK, LV=>LV, FV=>FV, DATIN=>DATIN, CLK25=>CLK25S, ENA_D=>ENA_DS,
IDLE=>IDLES,
    DQ_D=>DQ_DS, ADR_D=>ADR_DS, WE_D=>WE_DS);

U3:CTRL_UNIT
   PORT MAP
        (CLK25=>CLK25S, IDLE=>IDLES, PSH=>PSH, ENA_V=>ENA_VS, ENA_D=>ENA_DS);

U4:VGA3
        PORT MAP
   (CLK25=>CLK25S, ENA_V=>ENA_VS, REDOUT=>REDOUT,
   GREENOUT=>GREENOUT,CLKVGA=>CLKVGA,
    BLUEOUT=>BLUEOUT, H_SYNC_OUT=>H_SYNC_OUT,V_SYNC_OUT=>V_SYNC_OUT,
    BLANK=>BLANK, SYNC=>SYNC, WE_V=>WE_VS, DQ_V=>DQ_VS, ADR_V=>ADR_VS);

U5:BUSCTRL
        PORT MAP
        (SRAM_ADR=>SRAM_ADR, SRAM_DQ=>SRAM_DQ, ADR_D=>ADR_DS,
        ADR_V=>ADR_VS, DQ_V=>DQ_VS, DQ_D=>DQ_DS,
         IDLE=>IDLES, WE_D=>WE_DS, WE_V=>WE_VS, WE=>WE, CE=>CE, OE=>OE,
        UB=>UB, LB=>LB);
END a;
```

“ARITBILINEAL.vhd”

```
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ARITBILINEAL IS
        PORT
        (CLK            : IN    STD_LOGIC;
         REN,COL        : IN    STD_LOGIC_VECTOR(3 DOWNTO 0);
         PIXEL          : IN    STD_LOGIC_VECTOR(9 DOWNTO 0);
         RED,GREN,BLUE  : OUT STD_LOGIC_VECTOR(9 DOWNTO 0));
END ARITBILINEAL;


ARCHITECTURE a OF ARITBILINEAL IS
        SIGNAL INDX  : INTEGER range 0 to 13:=0 ;
        SIGNAL CONTREN,CONTCOL,R,C : INTEGER range 0 to 4 ;
        TYPE TABLE IS ARRAY (0 TO 4, 0 TO 4) OF INTEGER RANGE 0 TO 1023;
        SIGNAL T:TABLE;
        SIGNAL REDS,BLUES,GRENS:INTEGER range 0 to 1023 ;
BEGIN

T(0,0)<=50;T(0,1)<=50;T(0,2)<=25;T(0,3)<=35;T(0,4)<=10;
T(1,0)<=25;T(1,1)<=30;T(1,2)<=40;T(1,3)<=20;T(1,4)<=30;
T(2,0)<=40;T(2,1)<=40;T(2,2)<=30;T(2,3)<=35;T(2,4)<=20;
T(3,0)<=35;T(3,1)<=50;T(3,2)<=60;T(3,3)<=40;T(3,4)<=50;
T(4,0)<=10;T(4,1)<=55;T(4,2)<=25;T(4,3)<=50;T(4,4)<=40;

R<=CONV_INTEGER(REN);
C<=CONV_INTEGER(COL);
RED<=CONV_STD_LOGIC_VECTOR(REDS,10);
BLUE<=CONV_STD_LOGIC_VECTOR(BLUES,10);
GREN<=CONV_STD_LOGIC_VECTOR(GRENS,10);

PROCESS
BEGIN
        WAIT UNTIL CLK'EVENT AND CLK='1';

--Si es REN Par y COL Impar (Cuando esta en un Rojo)
        IF REN(0)='0' AND COL(0)='1' THEN
                REDS<=T(R,C);
                BLUES<=(T(R-1,C-1)+ T(R-1,C+1)+ T(R+1,C-1)+ T(R+1,C+1))/4;
                GRENS<=(T(R,C-1)+T(R-1,C)+T(R,C+1)+T(R+1,C))/4;

--Si es REN Impar y COL Par (Cuando esta en un Azul)
        ELSIF REN(0)='1' AND COL(0)='0' THEN
                BLUES<=T(R,C);
                REDS<=(T(R-1,C-1) + T(R-1,C+1) + T(R+1,C-1) + T(R+1,C+1))/4;
                GRENS<=(T(R,C-1) + T(R-1,C) + T(R,C+1) + T(R+1,C))/4;
```

“ARITBILINEAL.vhd” Continuación

```
--Cualquier otro caso (Cuando esta en un Verde)
        ELSE
                IF REN(0)='1' THEN --(Cuando esta renglon Impar)
                        GRENS<=T(R,C);
                        REDS<= (T(R-1,C) + T(R+1,C))/2;
                        BLUES<=(T(R,C-1) + T(R,C+1))/2;
                ELSE                --(Renglon par)
                        GRENS<=T(R,C);
                        BLUES<=(T(R-1,C) + T(R+1,C))/2;
                        REDS<=(T(R,C-1) + T(R,C+1))/2;

                END IF;

        END IF;

END PROCESS;

END a;
```

“PIPE.vhd”

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE WORK.CTES.ALL;
USE WORK.PAQUETE.ALL;

ENTITY PIPE IS
--GENERIC (NBITS : POSITIVE := 10;   --Ancho
--                  NTAP  : POSITIVE := 2 ;   --taps
--                  NWORD : POSITIVE :=640); --Longitud

                  PORT (
                  CLK : IN  STD_LOGIC;
                  ENA : IN  STD_LOGIC;
                  INP : IN  STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0);
                  TAP0,
                  TAP : OUT STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0);
                  OUTP: OUT STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0));
END PIPE;

ARCHITECTURE arch OF PIPE IS
TYPE LONG IS ARRAY (NTAP*NWORD DOWNTO 0) OF STD_LOGIC_VECTOR(NBITS-1
DOWNTO 0);
SIGNAL PIP: LONG;

BEGIN
PROCESS (CLK)
BEGIN
IF (CLK'EVENT AND CLK='0') THEN
        IF (ENA= '1') THEN
        PIP(NTAP*NWORD DOWNTO 1) <= PIP(NTAP*NWORD-1 DOWNTO 0);
        PIP(0) <=INP;
        END IF;
END IF;
END PROCESS;
--TAP<=PIP(((NTAP-1)*NWORD)-1);
TAP<=PIP(NWORD);
OUTP<=PIP(NTAP*NWORD);
TAP0<=PIP(0);
END arch;
```

“CONTPOS2.vhd”

```
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.CTES.ALL;
USE WORK.PAQUETE.ALL;

ENTITY CONTPOS2 IS
--GENERIC (NWORD : POSITIVE :=640; --Debe ser numero PAR de cols y rens debido al sensor
--                 MLINE : POSITIVE :=480;
--                 NBITS   : POSITIVE :=10);

PORT ( CLK                 :IN  STD_LOGIC;
     ENA                   :IN STD_LOGIC;
     ENAD                  :OUT STD_LOGIC;
     CONTYO,CONTCOL  :OUT INTEGER RANGE 0 TO NWORD-1;
     CONTXO,CONTREN  :OUT INTEGER RANGE 0 TO MLINE-1);
END CONTPOS2;


ARCHITECTURE a OF CONTPOS2 IS
          SIGNAL CONTX,CONTXO2       :INTEGER RANGE 0 TO MLINE-1;
          SIGNAL CONTY,CONTYO2       :INTEGER RANGE 0 TO NWORD-1;
          SIGNAL CONT                :INTEGER RANGE 0 TO NWORD+2;
          SIGNAL UNO                 :INTEGER RANGE 0 TO 1;

BEGIN
CONTXO<=CONTX;
CONTYO<=CONTY;
CONTREN<=CONTXO2;
CONTCOL<=CONTYO2;

POSICION:PROCESS(CLK)
BEGIN
IF CLK'EVENT AND CLK='1' THEN
          IF ENA='1' THEN
                    IF UNO=1 THEN
                              IF CONTY=NWORD-1 THEN
                              CONTY<=0;
                                        IF CONTX=MLINE-1 THEN
                                        CONTX<=0;
                                        ELSE
                                        CONTX<=CONTX+1;
                                        END IF;
                              ELSE
                              CONTY<=CONTY+1;
                              END IF;
```

“CONTPOS2.vhd” Continuación

```
                        IF CONT=NWORD+2 THEN
                                ENAD<='1';
                                IF CONTYO2=NWORD-1 THEN
                                CONTYO2<=0;
                                        IF CONTXO2=MLINE-1 THEN
                                        CONTXO2<=0;
                                        ELSE
                                        CONTXO2<=CONTXO2+1;
                                        END IF;
                                ELSE
                                CONTYO2<=CONTYO2+1;
                                END IF;
                ELSE
                        ENAD<='0';
                        CONT<=CONT+1;
                        END IF;
                ELSE
                UNO<=UNO+1;
                END IF;
        END IF;
END IF;
END PROCESS POSICION;

END a;
```

“PIPECONV2.vhd”

```
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.CTES.ALL;
USE WORK.PAQUETE.ALL;


ENTITY PIPECONV2 IS
--GENERIC (NWORD  : POSITIVE :=640;--Debe ser numero PAR de cols y rens debido al sensor
--              MLINE  : POSITIVE :=480;
--              NBITS   : POSITIVE :=10;
--              M         : POSITIVE :=3 );

PORT (  CLK :IN  STD_LOGIC;
        ENA             :IN       STD_LOGIC;
        CONTREN         :IN  INTEGER RANGE 0 TO MLINE-1;
        CONTCOL         :IN  INTEGER RANGE 0 TO NWORD-1;
        RED,GREN,BLUE      :OUT STD_LOGIC_VECTOR (NBITS-1 DOWNTO 0);
        INP,TAP,OUTP          :IN  STD_LOGIC_VECTOR (NBITS-1 DOWNTO 0));
END PIPECONV2;


ARCHITECTURE a OF PIPECONV2 IS
        TYPE CONV  IS ARRAY(0 TO M-1)OF INTEGER RANGE 0 TO 1023;
        SIGNAL C1,C2,C3 :CONV;
        SIGNAL REN,COL :STD_LOGIC_VECTOR (NBITS-1 DOWNTO 0);
        SIGNAL INW,TAPW,OUTW,REDS,BLUES,GRENS :INTEGER RANGE 0 TO 1023 ;
        SIGNAL UNO              :INTEGER RANGE 0 TO 1;

BEGIN
        RED  <=CONV_STD_LOGIC_VECTOR(REDS,NBITS);
        BLUE <=CONV_STD_LOGIC_VECTOR(BLUES,NBITS);
        GREN <=CONV_STD_LOGIC_VECTOR(GRENS,NBITS);
        REN  <=CONV_STD_LOGIC_VECTOR(CONTREN,NBITS);
        COL  <=CONV_STD_LOGIC_VECTOR(CONTCOL,NBITS);
        INW  <=CONV_INTEGER(INP);
        TAPW <=CONV_INTEGER(TAP);
        OUTW <=CONV_INTEGER(OUTP);


PROCESS(CLK)
BEGIN
IF (CLK'EVENT AND CLK='1') THEN
  IF ENA='1' THEN

        C1(0)<=INW;
        FOR J IN 1 TO M-1 LOOP
        C1(J)<=C1(J-1);
        END LOOP;

        C2(0)<=TAPW;
        FOR J IN 1 TO M-1 LOOP
        C2(J)<=C2(J-1);
        END LOOP;
```

“PIPECONV2.vhd” Continuación

```
            C3(0)<=OUTW;
            FOR J IN 1 TO M-1 LOOP
            C3(J)<=C3(J-1);
            END LOOP;

            END IF;
  END IF;
END PROCESS;


OPERACION:PROCESS
BEGIN
            WAIT UNTIL CLK'EVENT AND CLK='0';
            IF ENA='1' THEN
IF UNO=1 THEN
--Si es REN Par y COL Impar (Cuando esta en un ROJO)--
IF REN(0)='0' AND COL(0)='1' THEN
            --Si esta en el primer renglon y ultima columna (0,N-1)--
            IF REN=0 AND COL=NWORD-1 THEN
                        REDS<=C2(1);
            GRENS<=(C2(2)+C1(1))/4;
            BLUES<=C1(2)/4;
            --Si esta en el primer renglon--
            ELSIF REN=0 THEN
                        REDS<=C2(1);
                        GRENS<=(C2(2)+C2(0)+C1(1))/4;
                        BLUES<=(C1(2)+C1(0))/4;
             --Si esta en la ultima columna--
            ELSIF COL=NWORD-1 THEN
               REDS<=C2(1);
                        GRENS<=(C2(2)+C3(1)+C1(1))/4;
                        BLUES<=(C3(2)+C1(2))/4;
            --Cualquier otro caso--
            ELSE
               REDS<=C2(1);
      GRENS<=(C2(2)+C3(1)+C2(0)+C1(1))/4;
      BLUES<=(C3(2)+C3(0)+C1(2)+C1(0))/4;
   END IF;

--Si es REN Impar y COL Par (Cuando esta en un AZUL)--
ELSIF REN(0)='1' AND COL(0)='0' THEN
            --Si esta en el ultimo renglon y en la primera columna (N-1,0)--
            IF REN=MLINE-1 AND COL=0 THEN
                        REDS<=C3(0)/4;
                        GRENS<=(C3(1)+C2(0))/4;
                        BLUES<=C2(1);
            --Si esta en la primera columna--
            ELSIF COL=0 THEN
                        REDS<=(C3(0)+C1(0))/4;
                        GRENS<=(C3(1)+C2(0)+C1(1))/4;
                        BLUES<=C2(1);
```

“PIPECONV2.vhd” Continuación

```
        --Si esta en el ultimo renglon--
        ELSIF REN=MLINE-1 THEN
                REDS<=(C3(2)+C3(0))/4;
                GRENS<=(C2(2)+C3(1)+C2(0))/4;
                BLUES<=C2(1);
        --Cualquier otro caso--
        ELSE
                REDS<=(C3(2)+C3(0)+C1(2)+C1(0))/4;
                GRENS<=(C2(2)+C3(1)+C2(0)+C1(1))/4;
                BLUES<=C2(1);
        END IF;
--Cualquier otro caso (Cuando esta en un VERDE)
ELSE
        --Si esta en renglon Par--
        IF REN(0)='0' THEN
                --Si esta en el primer renglon y primera columna (0,0)--
                IF REN=0 AND COL=0 THEN
                REDS<=C2(0)/2;
                GRENS<=C2(1);
                BLUES<=C1(1)/2;
                --Si esta en el primer renglon--
                ELSIF REN=0 THEN
                REDS<=(C2(2)+C2(0))/2;
                GRENS<=C2(1);
                BLUES<=C1(1)/2;
                --Si esta en la primera columna--
                ELSIF COL=0 THEN
                REDS<=C2(0)/2;
                GRENS<=C2(1);
                BLUES<=(C3(1)+C1(1))/2;
                --Cualquier otro caso--
                ELSE
                REDS<=(C2(2)+C2(0))/2;
                GRENS<=C2(1);
                BLUES<=(C3(1)+C1(1))/2;
                END IF;

        --Si esta en renglon Impar--
        ELSE
                --Si esta en el ultimo renglon y la ultima columna (N-1,N-1)--
                IF REN=MLINE-1 AND COL=NWORD-1 THEN
                REDS<=C3(1)/2;
        GRENS<=C2(1);
                BLUES<=C2(2)/2;
                --Si esta en el ultimo renglon--
                ELSIF REN=MLINE-1 THEN
                REDS<=C3(1)/2;
        GRENS<=C2(1);
                BLUES<=(C2(2)+C2(0))/2;
                --Si esta en la ultima columna--
                ELSIF COL=NWORD-1 THEN
                REDS<= (C3(1)+C1(1))/2;
        GRENS<=C2(1);
                BLUES<=C2(2)/2;
```

“PIPECONV2.vhd” Continuación

```
                --Cualquier otro caso--
                ELSE
                REDS<= (C3(1)+C1(1))/2;
        GRENS<=C2(1);
                BLUES<=(C2(2)+C2(0))/2;
                END IF;
        END IF;
 END IF;
                ELSE
                UNO<=UNO+1;
                END IF;

END IF;
END PROCESS OPERACION;
END a;
```

“FRAMEBEGIN.vhd”

```
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY FRAMEBEGIN IS
PORT ( CLK,START         :IN  STD_LOGIC;
       LV,FV             :IN   STD_LOGIC;
       ENAFVLV,
       FVSIGO            :OUT STD_LOGIC);
END FRAMEBEGIN;

ARCHITECTURE a OF FRAMEBEGIN IS
       TYPE TIPOESTADO IS (EDOA,EDOB,EDOC);
       SIGNAL ESTADO:TIPOESTADO;
       SIGNAL FVSIG  :STD_LOGIC;
BEGIN
FVSIGO<=FVSIG;
ENAFVLV<=LV AND FVSIG;
PROCESS(CLK)--,ENA)
BEGIN
       IF CLK'EVENT AND CLK='1' THEN
              IF START='1' THEN
                     CASE ESTADO IS
                            WHEN EDOA =>
                        IF FV='0' THEN
                                   ESTADO<=EDOB;
                            ELSE
                                   ESTADO<=EDOA;
                            END IF;
                           WHEN EDOB =>
                         IF FV='1' THEN
                            ESTADO<=EDOC;
                            ELSE
                            ESTADO<=EDOB;
                         END IF;
                           WHEN EDOC =>
                         IF FV='0' THEN
                            ESTADO<=EDOA;
                            ELSE
                            ESTADO<=EDOC;
                         END IF;

                     END CASE;
              END IF;
       END IF;
END PROCESS FRAMEBEGIN;

WITH ESTADO SELECT
        FVSIG<='0' WHEN EDOA,
                     '0' WHEN EDOB,
       '1' WHEN EDOC;

END a;
```

“PIPEBILINEAL.vhd”

```
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.PAQUETE.ALL;
USE WORK.CTES.ALL;

ENTITY PIPEBILINEAL IS
--GENERIC (NWORD  : POSITIVE :=640; --Debe ser numero PAR de cols y rens debido al sensor
--              MLINE    : POSITIVE :=480;
--              NTAP     : POSITIVE := 2;
--              NBITS    : POSITIVE :=10);

PORT ( PB,  PXCLK         :IN    STD_LOGIC;
     FV,LV                :IN    STD_LOGIC;
     ENAD,ENAFVLV         :OUT STD_LOGIC;
     INWORD               :IN    STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0);
    RED,GREEN,BLUE        :OUT STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0));
END PIPEBILINEAL;

ARCHITECTURE a OF PIPEBILINEAL IS
          SIGNAL ENAS,STARTS      : STD_LOGIC;
          SIGNAL CONTRENS   :INTEGER RANGE 0 TO MLINE-1;
          SIGNAL CONTCOLS   :INTEGER RANGE 0 TO NWORD-1;
          SIGNAL TAPS,TAP0S,OUTPS :STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0);
BEGIN
ENAFVLV<=ENAS;

U0:PSH
          PORT MAP
          (PB=>PB, CLK25=>PXCLK, START=>STARTS);


U1:FRAMEBEGIN
          PORT MAP
          (CLK=>PXCLK, FV=>FV, LV=>LV, ENAFVLV=>ENAS, START=>STARTS );

U2:CONTPOS2
          PORT MAP
          (CLK=>PXCLK, ENA=>ENAS, CONTREN=>CONTRENS, CONTCOL=>CONTCOLS,
ENAD=>ENAD);

U3:PIPE
          PORT MAP
          (INP=>INWORD, CLK=>PXCLK, ENA=>ENAS, TAP=>TAPS, TAP0=>TAP0S,
OUTP=>OUTPS);

U4:PIPECONV2
          PORT MAP
          (INP=>TAP0S, TAP=>TAPS, OUTP=>OUTPS, CONTREN=>CONTRENS,
CONTCOL=>CONTCOLS, ENA=>ENAS,
           CLK=>PXCLK, RED=>RED, GREN=>GREEN, BLUE=>BLUE);

END a;
```

“BUFDESP3.vhd”

```
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY BUFDESP3 IS
        GENERIC (NWORD : POSITIVE :=640 );

PORT ( PXCLK,CLK25,
     FV,LV,VGA_ON       :IN STD_LOGIC;
          START          :IN STD_LOGIC;
          ENA_PX,
          ENAVGA        :OUT    STD_LOGIC;
          CONTWR        :OUT    INTEGER RANGE 0 TO 2*NWORD-1;
          CONTRD        :OUT    INTEGER RANGE 0 TO 2*NWORD-1 );
END BUFDESP3;

ARCHITECTURE a OF BUFDESP3 IS
        SIGNAL CONTWRS  :INTEGER RANGE 0 TO 2*NWORD-1;
        SIGNAL CONTRDS  :INTEGER RANGE 0 TO 2*NWORD-1;
   SIGNAL CONT     :INTEGER RANGE 0 TO NWORD;
   SIGNAL RETA     :INTEGER RANGE 0 TO 1;
        SIGNAL FVSIG    :STD_LOGIC;
        TYPE TIPOESTADO IS (EDOA,EDOB,EDOC);
        SIGNAL ESTADO:TIPOESTADO;
BEGIN

CONTWR<=CONTWRS;
CONTRD<=CONTRDS;
ENA_PX<=FVSIG AND LV;

CUENTA:PROCESS
BEGIN
        WAIT UNTIL PXCLK'EVENT AND PXCLK='1';
IF START='1' THEN
        IF FVSIG='1' AND LV='1' THEN

                                IF CONTWRS=2*NWORD-1 THEN
                                        CONTWRS<=0;
                                ELSE
                                        CONTWRS<=CONTWRS+1;
                                END IF;

                                IF CONT=NWORD THEN
                                        ENAVGA<='1';          --ACTIVA EL DESPLIEGUE VGA
                                ELSE
                                        CONT<=CONT+1;
                                        ENAVGA<='0';
                                END IF;
        END IF;
END IF;
END PROCESS CUENTA ;
```

“BUFDESP3.vhd” Continuacion

```
CUENTA2:PROCESS
BEGIN
        WAIT UNTIL CLK25'EVENT AND CLK25='0';
IF START='1' THEN
                IF VGA_ON='1' THEN
                    --EL TIEMPO VALIDO DE DESPLIEGUE (VIDEO_ON='1') DEL VGA--
                        IF RETA=1 THEN
                                IF CONTRDS=2*NWORD-1 THEN
                                CONTRDS<=0;
                                ELSE
                                CONTRDS<=CONTRDS+1;
                                END IF;
                        ELSE
                          RETA<=RETA+1;
                        END IF;

                END IF;
END IF;
END PROCESS CUENTA2 ;

FRAMEBEGIN:PROCESS(PXCLK)
BEGIN
        IF PXCLK'EVENT AND PXCLK='1' THEN
IF START='1' THEN
                        CASE ESTADO IS
                                WHEN EDOA =>
                           IF FV='0' THEN
                                        ESTADO<=EDOB;
                                 ELSE
                                        ESTADO<=EDOA;
                                 END IF;
                                WHEN EDOB =>
                             IF FV='1' THEN
                                  ESTADO<=EDOC;
                                  ELSE
                                  ESTADO<=EDOB;
                             END IF;

                                WHEN EDOC =>
                             IF FV='0' THEN
                                  ESTADO<=EDOA;
                                  ELSE
                                  ESTADO<=EDOC;
                             END IF;
                        END CASE;
END IF;
        END IF;
END PROCESS FRAMEBEGIN;

WITH ESTADO SELECT
         FVSIG<='0' WHEN EDOA,
                        '0' WHEN EDOB,
        '1' WHEN EDOC;
END a;
```

“BUFLINE2.vhd”

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY BUFLINE2 IS
GENERIC
        (NWORD   : POSITIVE:=640;
        NBITS    : POSITIVE:=10 );
 PORT
  (  CLKWR                          : IN   std_logic;
     CLKRD                          : IN   std_logic;
     INWORD                         : IN   std_logic_vector(NBITS-1 downto 0);
     WRADR                          : IN   INTEGER RANGE 0 TO 2*NWORD-1;
     RDADR                          : IN   INTEGER RANGE 0 TO 2*NWORD-1;
     WRENA,RDENA                    : IN   std_logic;
     OUTWORD                        : OUT  std_logic_vector(NBITS-1 downto 0) );
END BUFLINE2;

ARCHITECTURE rtl OF BUFLINE2 IS
TYPE RAM IS ARRAY(0 TO 2*NWORD-1 ) of std_logic_vector(NBITS-1 downto 0);
SIGNAL BUF :RAM;
  BEGIN

ESCRITURA:PROCESS (CLKWR)
  BEGIN
 IF (CLKWR'event AND CLKWR = '0') THEN
        IF (WRENA='1') THEN
      BUF(WRADR) <=INWORD;
        END IF;
 END IF;
END PROCESS ESCRITURA;

LECTURA:PROCESS (CLKRD)
  BEGIN
  IF (CLKRD'event AND CLKRD = '1') THEN
        IF(RDENA='1') THEN
                OUTWORD <= BUF(RDADR);
    END IF;
  END IF;
END PROCESS LECTURA;
END rtl;
```

"VGA4.vhd"

```
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY VGA4 IS
        PORT
        (CLK25,ENA_V                    :IN STD_LOGIC;
         REDOUT,GREENOUT,BLUEOUT        :OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
         H_SYNC_OUT,V_SYNC_OUT,
         VIDEO_ON,
         BLANK,SYNC,CLKVGA              :OUT STD_LOGIC;
          DQ_V                          :IN STD_LOGIC_VECTOR(9 DOWNTO 0));
END VGA4;

ARCHITECTURE a OF VGA4 IS
        SIGNAL H_SYNC,V_SYNC            :STD_LOGIC;
        SIGNAL VIDEO_ONS,VIDEO_ON_V,VIDEO_ON_H :STD_LOGIC;
        SIGNAL H_COUNT,V_COUNT          :STD_LOGIC_VECTOR(9 DOWNTO 0);
--      SIGNAL COUNT_ADR                :STD_LOGIC_VECTOR(17 DOWNTO 0);
--      SIGNAL GREENS                   :STD_LOGIC_VECTOR(9 DOWNTO 0);
BEGIN

--Reloj para el DAC VGA
CLKVGA<=CLK25;
--La señal VIDEO_ON es equivalente a BLANK del DAC de video ADV7123
VIDEO_ONS<=VIDEO_ON_H AND VIDEO_ON_V;
--BLANK<=VIDEO_ON;
--Se pone esta salida para que sea el habilitador para el contador de lectura para el comp. BUFDESP3--
VIDEO_ON<=VIDEO_ONS;
 SYNC<='0';
--Se ponen a unos puesto que solo se utilizara el componente R( 8 bits msb solamente) del RGB
REDOUT<="0000000000";
BLUEOUT<="0000000000";
--Pone en GREENS lo que tiene en el puerto DQ_V,solo los 10
GREENOUT<=DQ_V;

PROCESS
BEGIN

WAIT UNTIL (CLK25'EVENT) AND (CLK25='1');
--Habilitador para el proceso y poner en Z EL SDRAM_DQ y WE,WE=1 pone a lectura a la SRAM
IF ENA_V='1' THEN

        --Generar las señales de tiempos Horizontal y Vertical para la señal de video
        --H_COUNT cuenta pixeles(640+tiempo extra para señales de sincronizacion)
        --
        --HORIZ_SYNC ----------_____------
        --H_COUNT    0       640       659    755   799

        IF (H_COUNT=799) THEN
        H_COUNT<="0000000000";
        ELSE
        H_COUNT<=H_COUNT+1;
        END IF;
```

“VGA4.vhd” Continuacion

```
        --Generar Señal de sincronizacion Horizontal usando H_COUNT--
        IF (H_COUNT<=755) AND (H_COUNT>=659) THEN
                H_SYNC<='0';
        ELSE
           H_SYNC<='1';
        END IF;
        --V_COUNT cuenta renglones de pixeles (480+ tiempo extra para señales de sincronizacion)
        --VERT_SYNC ---------------------------_______---------
        --V_COUNT   0          480          493-494      524

        IF (V_COUNT>=524) AND (H_COUNT>=799) THEN
        V_COUNT<="0000000000";
        ELSIF (H_COUNT=799) THEN
        V_COUNT<=V_COUNT+1;
        END IF;

        --Generar señal de sincronizacion Vertical usando V_COUNT
        IF (V_COUNT<=494) AND (V_COUNT>=493) THEN
                V_SYNC<='0';
        ELSE
                V_SYNC<='1';
        END IF;

        --Generar Señales de Video en Pantalla para los datos de los pixeles
        IF (H_COUNT<=639) THEN
                VIDEO_ON_H<='1';
        ELSE
                VIDEO_ON_H<='0';
        END IF;


        IF (V_COUNT<=479) THEN
           VIDEO_ON_V<='1';
        ELSE
                VIDEO_ON_V<='0';
        END IF;

        V_SYNC_OUT<=V_SYNC;
   H_SYNC_OUT<=H_SYNC;
        BLANK<=VIDEO_ONS;

END IF;

END PROCESS;
END a;
```

“PSH.vhd”

```
LIBRARY IEEE;
USE IEEE. STD_LOGIC_1164.ALL;
USE IEEE. STD_LOGIC_ARITH. ALL;

ENTITY PSH  IS
        PORT
        (CLK25          : IN  STD_LOGIC;
         PB             : IN  STD_LOGIC;
         START          : OUT STD_LOGIC);
END PSH;


ARCHITECTURE a OF PSH IS
        TYPE STATE_TYPE IS (A,B,C);
        SIGNAL state: STATE_TYPE;
BEGIN

        PROCESS (CLK25)
        BEGIN
        IF CLK25'EVENT AND CLK25='1' THEN
                        CASE state IS
                           WHEN A =>
                           IF PB='0' THEN
                             state<=B;
                                END IF;

                                WHEN B =>
                                        IF PB='1' THEN
                                                state <=C;
                                        END IF;
                                WHEN C=>
                                state <=C;

                        END CASE;
                END IF;
        END PROCESS;

        WITH state SELECT
                START <='0' WHEN A,
                                '0' WHEN B,
                                '1' WHEN C;
END a;
```

“DESP_REAL3.vhd”

```
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.PAQUETE.ALL;
ENTITY DESP_REAL3 IS
GENERIC
        (NWORD   : POSITIVE:=640;
        NBITS   : POSITIVE:=10 );
PORT
 (PB,PXCLK                :IN     STD_LOGIC;
 LV,FV                    :IN     STD_LOGIC;
START,
ENAVGA                    :OUT    STD_LOGIC;
RDENA,WRENA               :OUT    STD_LOGIC;
INWORD                    :IN     STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0);
REDOUT,GREENOUT,BLUEOUT   :OUT    STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0);
H_SYNC_OUT,V_SYNC_OUT,
BLANK,SYNC,CLKVGA         :OUT STD_LOGIC);
END DESP_REAL3;
ARCHITECTURE a OF DESP_REAL3 IS
        SIGNAL ENA_PXS, VGA_ONS, ENAVGAS, CLK25S, STARTS :STD_LOGIC;
        SIGNAL CONTWRS,CONTRDS  :INTEGER RANGE 0 TO 2*NWORD-1;
        SIGNAL OUTWORDS         :STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0);

BEGIN
WRENA<=ENA_PXS;
RDENA<=VGA_ONS;
ENAVGA<=ENAVGAS;
START<=STARTS;
U1:BUFDESP3
        PORT MAP
   (PXCLK=>PXCLK, LV=>LV, FV=>FV, CLK25=>CLK25S, ENAVGA=>ENAVGAS,
START=>STARTS,
    VGA_ON=>VGA_ONS, ENA_PX=>ENA_PXS, CONTWR=>CONTWRS, CONTRD=>CONTRDS
);
U2:BUFLINE2
        PORT MAP
        (CLKWR=>PXCLK, CLKRD=>CLK25S, INWORD=>INWORD, WRENA=>ENA_PXS,
RDENA=>VGA_ONS, OUTWORD=>OUTWORDS,
         WRADR=>CONTWRS, RDADR=>CONTRDS);
U3:VGA4
        PORT MAP
        (CLK25=>CLK25S, ENA_V=>ENAVGAS,  REDOUT=>REDOUT,
GREENOUT=>GREENOUT ,BLUEOUT=>BLUEOUT, VIDEO_ON=>VGA_ONS,
          H_SYNC_OUT=>H_SYNC_OUT,  V_SYNC_OUT=>V_SYNC_OUT, BLANK=>BLANK
,SYNC=>SYNC, CLKVGA=>CLKVGA,
          DQ_V=>OUTWORDS  );
U4:CLKDIVIDE
        PORT MAP
        (CLK50MHZ=>PXCLK, CLK25MHZ=>CLK25S);

U5:PSH
        PORT MAP
        (CLK25=>CLK25S, PB=>PB, START=>STARTS);
END a;
```

“BUFDESP4.vhd”

```
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.CTES.ALL;
USE WORK.PAQUETE.ALL;

ENTITY BUFDESP4 IS
--          GENERIC (NWORD : POSITIVE :=640 );

PORT (
PXCLK,CLK25,
VGA_ON              :IN STD_LOGIC;
ENAD,ENAFVLV :IN STD_LOGIC;
WRENA,
ENAVGA            :OUT STD_LOGIC;
CONTWR          :OUT INTEGER RANGE 0 TO 2*NWORD-1;
 CONTRD          :OUT INTEGER RANGE 0 TO 2*NWORD-1 );

END BUFDESP4;


ARCHITECTURE a OF BUFDESP4 IS
SIGNAL CONTWRS  :INTEGER RANGE 0 TO 2*NWORD-1;
SIGNAL CONTRDS  :INTEGER RANGE 0 TO 2*NWORD-1;
SIGNAL CONT     :INTEGER RANGE 0 TO NWORD;
SIGNAL RETA     :INTEGER RANGE 0 TO 1;

BEGIN

CONTWR<=CONTWRS;
CONTRD<=CONTRDS;
WRENA<=ENAD AND ENAFVLV;

CUENTA:PROCESS
BEGIN
          WAIT UNTIL PXCLK'EVENT AND PXCLK='1';
IF ENAD='1' THEN
          IF ENAFVLV='1' THEN

                                        IF CONTWRS=2*NWORD-1 THEN
                                                  CONTWRS<=0;
                                        ELSE
                                                  CONTWRS<=CONTWRS+1;
                                        END IF;

                                        IF CONT=NWORD THEN
                                                  ENAVGA<='1';          --ACTIVA EL DESPLIEGUE VGA
                                        ELSE
                                                  CONT<=CONT+1;
                                                  ENAVGA<='0';
                                        END IF;
          END IF;
END IF;
END PROCESS CUENTA ;
```

“BUFDESP4.vhd” Continuacion

```
CUENTA2:PROCESS
BEGIN
        WAIT UNTIL CLK25'EVENT AND CLK25='0';
IF ENAD='1' THEN
                IF VGA_ON='1' TH
--EL TIEMPO VALIDO DE DESPLIEGUE (VIDEO_ON='1') DEL VGA--
                        IF RETA=1 THEN
                                IF CONTRDS=2*NWORD-1 THEN
                                CONTRDS<=0;
                                ELSE
                                CONTRDS<=CONTRDS+1;
                                END IF;
                        ELSE
                          RETA<=RETA+1;
                        END IF;

                END IF;
END IF;
END PROCESS CUENTA2 ;
END a;
```

“BUFRED.vhd”

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.CTES.ALL;
USE WORK.PAQUETE.ALL;

ENTITY BUFRED IS
--GENERIC
--        (       NWORD   : POSITIVE:=640;
--                NBITS     : POSITIVE:=10);
 PORT
  (  CLKWR                                          : IN   std_logic;
     CLKRD                                          : IN   std_logic;
     INRED                                          : IN   std_logic_vector(NBITS-1 downto 0);
     WRADR                                          : IN   INTEGER RANGE 0 TO 2*NWORD-1;
     RDADR                                          : IN   INTEGER RANGE 0 TO 2*NWORD-1;
     WRENA,RDENA                                    : IN   std_logic;
     OUTRED                                         : OUT  std_logic_vector(NBITS-1 downto 0) );
END BUFRED;

ARCHITECTURE rtl OF BUFRED IS
TYPE RAM IS ARRAY(0 TO 2*NWORD-1 ) of std_logic_vector(NBITS-1 downto 0);
SIGNAL BUF :RAM;
  BEGIN

ESCRITURA:PROCESS (CLKWR)
  BEGIN
 IF (CLKWR'event AND CLKWR = '0') THEN
          IF (WRENA='1') THEN
      BUF(WRADR) <=INRED;
          END IF;
 END IF;
END PROCESS ESCRITURA;

LECTURA:PROCESS (CLKRD)
  BEGIN
  IF (CLKRD'event AND CLKRD = '1') THEN
          IF(RDENA='1') THEN
                  OUTRED <= BUF(RDADR);
    END IF;
  END IF;
END PROCESS LECTURA;

END rtl;
```

“BUFGREEN.vhd”

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.CTES.ALL;
USE WORK.PAQUETE.ALL;

ENTITY BUFGREEN IS
--GENERIC
--          (          NWORD   : POSITIVE:=640;
--                     NBITS   : POSITIVE:=10 );
 PORT
   ( CLKWR                                          : IN    std_logic;
     CLKRD                                          : IN    std_logic;
     INGREEN                                                :IN    std_logic_vector(NBITS-1 downto
0);
     WRADR                                          : IN    INTEGER RANGE 0 TO 2*NWORD-1;
     RDADR                                          : IN    INTEGER RANGE 0 TO 2*NWORD-1;
     WRENA,RDENA                                    : IN    std_logic;
     OUTGREEN                                       : OUT   std_logic_vector(NBITS-1 downto 0) );
END BUFGREEN;

ARCHITECTURE rtl OF BUFGREEN IS
TYPE RAM IS ARRAY(0 TO 2*NWORD-1 ) of std_logic_vector(NBITS-1 downto 0);
SIGNAL BUF :RAM;
  BEGIN

ESCRITURA:PROCESS (CLKWR)
  BEGIN
 IF (CLKWR'event AND CLKWR = '0') THEN
          IF (WRENA='1') THEN
       BUF(WRADR) <=INGREEN;
          END IF;
 END IF;
END PROCESS ESCRITURA;

LECTURA:PROCESS (CLKRD)
  BEGIN
  IF (CLKRD'event AND CLKRD = '1') THEN
          IF(RDENA='1') THEN
                    OUTGREEN <= BUF(RDADR);
    END IF;
  END IF;
END PROCESS LECTURA;

END rtl;
```

“BUFBLUE.vhd”

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.CTES.ALL;
USE WORK.PAQUETE.ALL;

ENTITY BUFBLUE IS
--GENERIC
--        (       NWORD   : POSITIVE:=640;
--                NBITS   : POSITIVE:=10 );
 PORT
  ( CLKWR                                   : IN    std_logic;
    CLKRD                                   : IN    std_logic;
    INBLUE                                  : IN    std_logic_vector(NBITS-1 downto 0);
    WRADR                                   : IN    INTEGER RANGE 0 TO 2*NWORD-1;
    RDADR                                   : IN    INTEGER RANGE 0 TO 2*NWORD-1;
    WRENA,RDENA                             : IN    std_logic;
    OUTBLUE                                 : OUT   std_logic_vector(NBITS-1 downto 0) );
END BUFBLUE;

ARCHITECTURE rtl OF BUFBLUE IS
TYPE RAM IS ARRAY(0 TO 2*NWORD-1 ) of std_logic_vector(NBITS-1 downto 0);
SIGNAL BUF :RAM;
  BEGIN

ESCRITURA:PROCESS (CLKWR)
  BEGIN
 IF (CLKWR'event AND CLKWR = '0') THEN
          IF (WRENA='1') THEN
      BUF(WRADR) <=INBLUE;
          END IF;
 END IF;
END PROCESS ESCRITURA;

LECTURA:PROCESS (CLKRD)
  BEGIN
  IF (CLKRD'event AND CLKRD = '1') THEN
          IF(RDENA='1') THEN
                  OUTBLUE <= BUF(RDADR);
    END IF;
  END IF;
END PROCESS LECTURA;

END rtl;
```

“VGA5.vhd”

```
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.CTES.ALL;
USE WORK.PAQUETE.ALL;

ENTITY VGA5 IS
        PORT
        ( CLK25,ENA_V                       :IN STD_LOGIC;
          REDOUT,GREENOUT,BLUEOUT    :OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
          H_SYNC_OUT,V_SYNC_OUT,
          VIDEO_ON,
          BLANK,SYNC,CLKVGA         :OUT STD_LOGIC;
          REDIN,GREENIN,BLUEIN      :IN STD_LOGIC_VECTOR(9 DOWNTO 0));
END VGA5;

ARCHITECTURE a OF VGA5 IS
        SIGNAL H_SYNC,V_SYNC              :STD_LOGIC;
        SIGNAL VIDEO_ONS,VIDEO_ON_V,VIDEO_ON_H :STD_LOGIC;
        SIGNAL H_COUNT,V_COUNT            :STD_LOGIC_VECTOR(9 DOWNTO 0);
BEGIN

--Reloj para el DAC VGA
CLKVGA<=CLK25;
--La señal VIDEO_ON es equivalente a BLANK del DAC de video ADV7123
VIDEO_ONS<=VIDEO_ON_H AND VIDEO_ON_V;
--Se pone esta salida para que sea el habilitador para el contador de lectura para el comp. BUFDESP3--
VIDEO_ON<=VIDEO_ONS;
 SYNC<='0';

REDOUT<=REDIN;
GREENOUT<=GREENIN;
BLUEOUT<=BLUEIN;

PROCESS
BEGIN
WAIT UNTIL (CLK25'EVENT) AND (CLK25='1');
--Habilitador para el proceso y poner en Z EL SDRAM_DQ y WE,WE=1 pone a lectura a la SRAM
IF ENA_V='1' THEN

        --Generar las señales de tiempos Horizontal y Vertical para la señal de video
        --H_COUNT cuenta pixeles(640+tiempo extra para señales de sincronizacion)
        --
        --HORIZ_SYNC ----------_____-----
        --H_COUNT    0       640       659    755   799

        IF (H_COUNT=799) THEN
        H_COUNT<="0000000000";
        ELSE
        H_COUNT<=H_COUNT+1;
        END IF;
```

“VGA5.vhd” Continuación

```
--Generar Señal de sincronizacion Horizontal usando H_COUNT--
        IF (H_COUNT<=755) AND (H_COUNT>=659) THEN
        H_SYNC<='0';
        ELSE
        H_SYNC<='1';
        END IF;
        --V_COUNT cuenta renglones de pixeles (480+ tiempo extra para señales de sincronizacion)
        --VERT_SYNC -------------------------_______---------
        --V_COUNT   0          480          493-494     524

        IF (V_COUNT>=524) AND (H_COUNT>=799) THEN
        V_COUNT<="0000000000";
        ELSIF (H_COUNT=799) THEN
        V_COUNT<=V_COUNT+1;
        END IF;

        --Generar señal de sincronizacion Vertical usando V_COUNT
        IF (V_COUNT<=494) AND (V_COUNT>=493) THEN
                V_SYNC<='0';
        ELSE
                V_SYNC<='1';
        END IF;

        --Generar Señales de Video en Pantalla para los datos de los pixeles
        IF (H_COUNT<=639) THEN
                VIDEO_ON_H<='1';
        ELSE
                VIDEO_ON_H<='0';
        END IF;


        IF (V_COUNT<=479) THEN
           VIDEO_ON_V<='1';
        ELSE
                VIDEO_ON_V<='0';
        END IF;

        V_SYNC_OUT<=V_SYNC;
        H_SYNC_OUT<=H_SYNC;
        BLANK<=VIDEO_ONS;

END IF;
END PROCESS;
END a;
```

“DESP_REAL4.vhd”

```
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.PAQUETE.ALL;
USE WORK.CTES.ALL;

ENTITY DESP_REAL4 IS
--GENERIC
--          (          NWORD   : POSITIVE:=640;
--                     NBITS   : POSITIVE:=10 );
PORT
(ENAD,ENAFVLV,
PXCLK                              :IN     STD_LOGIC;
INRED,INGREEN,INBLUE         :IN     STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0);
REDOUT,GREENOUT,BLUEOUT      :OUT    STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0);
H_SYNC_OUT,V_SYNC_OUT,
BLANK,SYNC,CLKVGA,WRENA,RDENA,ENAVGA            :OUT STD_LOGIC);
END DESP_REAL4;

ARCHITECTURE a OF DESP_REAL4 IS
SIGNAL WRENAS, VGA_ONS, ENAVGAS, CLK25S :STD_LOGIC;
SIGNAL CONTWRS,CONTRDS                                  :INTEGER RANGE 0 TO 2*NWORD-1;
SIGNAL OUTREDS,OUTGREENS,OUTBLUES    :STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0);

BEGIN

WRENA<=WRENAS;
RDENA<=VGA_ONS;
ENAVGA<=ENAVGAS;

U1:BUFDESP4
          PORT MAP
   (PXCLK=>PXCLK, CLK25=>CLK25S, ENAVGA=>ENAVGAS, ENAD=>ENAD,
ENAFVLV=>ENAFVLV,
    VGA_ON=>VGA_ONS, WRENA=>WRENAS, CONTWR=>CONTWRS, CONTRD=>CONTRDS
);

U2:BUFRED
          PORT MAP
          (CLKWR=>PXCLK, CLKRD=>CLK25S, INRED=>INRED, WRENA=>WRENAS,
RDENA=>VGA_ONS, OUTRED=>OUTREDS,
           WRADR=>CONTWRS, RDADR=>CONTRDS);

U3:BUFGREEN
          PORT MAP
          (CLKWR=>PXCLK, CLKRD=>CLK25S, INGREEN=>INGREEN, WRENA=>WRENAS,
RDENA=>VGA_ONS, OUTGREEN=>OUTGREENS,
           WRADR=>CONTWRS, RDADR=>CONTRDS);

U4:BUFBLUE
          PORT MAP
          (CLKWR=>PXCLK, CLKRD=>CLK25S, INBLUE=>INBLUE, WRENA=>WRENAS,
RDENA=>VGA_ONS, OUTBLUE=>OUTBLUES,
           WRADR=>CONTWRS, RDADR=>CONTRDS);
```

“DESP_REAL4.vhd” Continuacion

```
U5:VGA5
        PORT MAP
        (CLK25=>CLK25S, ENA_V=>ENAVGAS,  REDOUT=>REDOUT,
GREENOUT=>GREENOUT ,BLUEOUT=>BLUEOUT, VIDEO_ON=>VGA_ONS,
          H_SYNC_OUT=>H_SYNC_OUT,  V_SYNC_OUT=>V_SYNC_OUT, BLANK=>BLANK
,SYNC=>SYNC, CLKVGA=>CLKVGA,
          REDIN=>OUTREDS, GREENIN=>OUTGREENS, BLUEIN=>OUTBLUES );

U6:CLKDIVIDE
        PORT MAP
        (CLK50MHZ=>PXCLK, CLK25MHZ=>CLK25S);

END a;
```

“MULTADD5.vhd”

```
LIBRARY ieee;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.CTES.ALL;
USE WORK.PAQUETE.ALL;

ENTITY MULTADD5 IS
PORT
(
R,G,B                   : IN    STD_LOGIC_VECTOR (9 DOWNTO 0);
RNO,GNO,BNO             : OUT   STD_LOGIC_VECTOR (9 DOWNTO 0);
CLK,ENAD,ENAFVLV        : IN    STD_LOGIC;
ENAM                    : OUT   STD_LOGIC);
END MULTADD5;

ARCHITECTURE rtl OF MULTADD5 IS
--Parte entera y frac. de los coeficientes--
CONSTANT R1I:UNSIGNED (1 DOWNTO 0):="01";                   -- 1
CONSTANT R1F:UNSIGNED (6 DOWNTO 0):="0101111";              --.37
CONSTANT R2I:UNSIGNED (1 DOWNTO 0):="00";                   -- 0
CONSTANT R2F:UNSIGNED (6 DOWNTO 0):="1000010";              --.52
CONSTANT R3I:UNSIGNED (1 DOWNTO 0):="00";                   -- 0
CONSTANT R3F:UNSIGNED (6 DOWNTO 0):="0000101";  --.04
CONSTANT G1I:UNSIGNED (1 DOWNTO 0):="00";                   -- 0
CONSTANT G1F:UNSIGNED (6 DOWNTO 0):="0110111";              --.43
CONSTANT G2I:UNSIGNED (1 DOWNTO 0):="01";                   -- 1
CONSTANT G2F:UNSIGNED (6 DOWNTO 0):="1000101";              --.54
CONSTANT G3I:UNSIGNED (1 DOWNTO 0):="00";                   -- 0
CONSTANT G3F:UNSIGNED (6 DOWNTO 0):="0001111";  --.12
CONSTANT B1I:UNSIGNED (1 DOWNTO 0):="00";                   -- 0
CONSTANT B1F:UNSIGNED (6 DOWNTO 0):="0100010";              --.27
CONSTANT B2I:UNSIGNED (1 DOWNTO 0):="01";                   -- 1
CONSTANT B2F:UNSIGNED (6 DOWNTO 0):="0100111";              --.31
CONSTANT B3I:UNSIGNED (1 DOWNTO 0):="10";                   -- 2
CONSTANT B3F:UNSIGNED (6 DOWNTO 0):="0111001";  --.45


SIGNAL RU,GU,BU                                         :UNSIGNED( 9 DOWNTO 0);
SIGNAL R1M,R2M,R3M,G1M,G2M,G3M,B1M,B2M,B3M               :UNSIGNED(16 DOWNTO 0);
SIGNAL R1T,R2T,R3T,G1T,G2T,G3T,B1T,B2T,B3T              :UNSIGNED(11 DOWNTO 0);
SIGNAL R1P,R2P,R3P,G1P,G2P,G3P,B1P,B2P,B3P              :UNSIGNED(11 DOWNTO 0);
SIGNAL R1S,R2S,R3S,G1S,G2S,G3S,B1S,B2S,B3S              :UNSIGNED(11 DOWNTO 0);
SIGNAL RN,GN,BN,RP,GP,BP                                :UNSIGNED(11 DOWNTO 0);
SIGNAL RETA                                             :INTEGER RANGE 0 TO 1;
SIGNAL RINT,GINT,BINT                                   :INTEGER RANGE 0 TO 1023;

BEGIN

RINT<=CONV_INTEGER(R); GINT<=CONV_INTEGER(G); BINT<=CONV_INTEGER(B);
RU<=CONV_UNSIGNED(RINT,10);
GU<=CONV_UNSIGNED(GINT,10);
BU<=CONV_UNSIGNED(BINT,10);
```

"MULTADD5.vhd" Continuacion

```
--Se toma la parte entera del resultado del producto de la parte frac. del coef. x el dato RGB--
R1T(9 DOWNTO 0)<=R1M(16 DOWNTO 7);
R2T(9 DOWNTO 0)<=R2M(16 DOWNTO 7);
R3T(9 DOWNTO 0)<=R3M(16 DOWNTO 7);
G1T(9 DOWNTO 0)<=G1M(16 DOWNTO 7);
G2T(9 DOWNTO 0)<=G2M(16 DOWNTO 7);
G3T(9 DOWNTO 0)<=G3M(16 DOWNTO 7);
B1T(9 DOWNTO 0)<=B1M(16 DOWNTO 7);
B2T(9 DOWNTO 0)<=B2M(16 DOWNTO 7);
B3T(9 DOWNTO 0)<=B3M(16 DOWNTO 7);

--Se rellena de 0's para tener el mismo numero de bits (12bits) y poder
--hacer la suma de la parte entera del resultado del producto de la parte frac. del coef. x el dato RGB +
--producto  de la parte ent. del coef. x el dato RGB
R1T(11 DOWNTO 10)<="00"; R2T(11 DOWNTO 10)<="00"; R3T(11 DOWNTO 10)<="00";
G1T(11 DOWNTO 10)<="00"; G2T(11 DOWNTO 10)<="00"; G3T(11 DOWNTO 10)<="00";
B1T(11 DOWNTO 10)<="00"; B2T(11 DOWNTO 10)<="00"; B3T(11 DOWNTO 10)<="00";

--Suma de la parte entera del resultado del producto de la (parte frac. del coef. x el dato RGB) +
--producto  de la (parte ent. del coef. x el dato RGB, (sumandos y resultado de 12 bits)
R1S<=R1T+R1P; R2S<=R2T+R2P; R3S<=R3T+R3P;
G1S<=G1T+G1P; G2S<=G2T+G2P; G3S<=G3T+G3P;
B1S<=B1T+B1P; B2S<=B2T+B2P; B3S<=B3T+B3P;
--Suma de las partes negativas de la matriz de correcion--
RP<=R2S+R3S;
GP<=G1S+G3S;
BP<=B1S+B2S;

RNO<=CONV_STD_LOGIC_VECTOR(RN,10); GNO<=CONV_STD_LOGIC_VECTOR(GN,10);
BNO<=CONV_STD_LOGIC_VECTOR(BN,10);

PROCESS (CLK)
BEGIN
IF (CLK'EVENT AND CLK='1') THEN
        IF ENAD='1' AND ENAFVLV='1' THEN
--producto parte frac. del coef. x el dato RGB--
R1M<=R1F*RU; R2M<=R2F*GU; R3M<=R3F*BU;
G1M<=G1F*RU; G2M<=G2F*GU; G3M<=G3F*BU;
B1M<=B1F*RU; B2M<=B2F*GU; B3M<=B3F*BU;
--producto parte ent. del coef. x el dato RGB--
R1P<=R1I*RU; R2P<=R2I*GU; R3P<=R3I*BU;
G1P<=G1I*RU; G2P<=G2I*GU; G3P<=G3I*BU;
B1P<=B1I*RU; B2P<=B2I*GU; B3P<=B3I*BU;

--ACONDICIONAMIENTO PARA LA SALIDA
--Si la parte negaitva es mayor que la positiva resultado RN a 0--
IF (RP>=R1S) THEN
        RN<="000000000000";
--Si es mayor que el maximo techarlo al maximo (1023)
ELSIF (RN>1023) THEN
        RN<="001111111111";
--Si no es ninguno de los casos anteriores hacer la suma ponderada pertinente
ELSE
        RN<=R1S-R2S-R3S;
END IF;
```

“MULTADD5.vhd” Continuación

```
IF (GP>=G2S) THEN
        GN<="000000000000";
ELSIF  (GN>1023) THEN
        GN<="001111111111";
ELSE
        GN<=G2S-G3S-G1S;
END IF;

IF (BP>=B3S) THEN
        BN<="000000000000";
ELSIF  (BN>1023) THEN
        BN<="001111111111";
ELSE
        BN<=B3S-B1S-B2S;
END     IF;

                IF RETA=1 THEN
                ENAM<='1';
                ELSE
                ENAM<='0';
                RETA<=RETA+1;
                END IF;

        END IF;
END IF;
END PROCESS;

END rtl;
```

“DESP_BAYER2.vhd”

```
LIBRARY IEEE,WORK;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.CTES.ALL;
USE WORK.PAQUETE.ALL;


ENTITY DESP_BAYER2 IS
PORT
 (PXCLK,FV,LV,PB                :IN  STD_LOGIC;
INWORD                          :IN  STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0);
REDOUT,GREENOUT,BLUEOUT  :OUT STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0);
ENAM,ENAD,ENAFVLV             :OUT STD_LOGIC;
H_SYNC_OUT,V_SYNC_OUT,
BLANK,SYNC,CLKVGA,WRENA,RDENA,ENAVGA   :OUT STD_LOGIC);
END DESP_BAYER2;

ARCHITECTURE a OF DESP_BAYER2 IS
SIGNAL ENADS,ENAFVLVS,CLK25S,ENAMS    :STD_LOGIC;
SIGNAL REDS,GREENS,BLUES                        :STD_LOGIC_VECTOR(NBITS-1
DOWNTO 0);
SIGNAL INREDS,INGREENS,INBLUES                  :STD_LOGIC_VECTOR(NBITS-1
DOWNTO 0);

BEGIN

ENAD<=ENADS; ENAFVLV<=ENAFVLVS;ENAM<=ENAMS;
--RN<=INREDS; GN<=INGREENS; BN<=INBLUES;

U1:PIPEBILINEAL
        PORT MAP
        (PXCLK=>PXCLK, FV=>FV, LV=>LV, PB=>PB, ENAD=>ENADS,
ENAFVLV=>ENAFVLVS,
   INWORD=>INWORD, RED=>REDS, GREEN=>GREENS, BLUE=>BLUES);

U2:MULTADD5
        PORT MAP
        (CLK=>PXCLK, ENAD=>ENADS, ENAFVLV=>ENAFVLVS, R=>REDS,G=>GREENS,
B=>BLUES, RNO=>INREDS,
         GNO=>INGREENS, BNO=>INBLUES,ENAM=>ENAMS );


U3:DESP_REAL4
        PORT MAP
        (ENAD=>ENAMS, ENAFVLV=>ENAFVLVS, PXCLK=>PXCLK, INRED=>INREDS,
INGREEN=>INGREENS, INBLUE=>INBLUES,
         CLKVGA=>CLKVGA, REDOUT=>REDOUT, GREENOUT=>GREENOUT,
BLUEOUT=>BLUEOUT, H_SYNC_OUT=>H_SYNC_OUT,
         V_SYNC_OUT=>V_SYNC_OUT, BLANK=>BLANK, SYNC=>SYNC, WRENA=>WRENA,
ENAVGA=>ENAVGA, RDENA=>RDENA);

END a;
```

Apéndice C Paquetes de declaraciones de componentes y de parámetros configurables. (PAQUETE.vhd y CTES.vhd)

"PAQUETE.vhd"

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE WORK.CTES.ALL;
USE WORK.PAQUETE.ALL;

PACKAGE PAQUETE IS

COMPONENT CLKDIV
PORT
(CLK25MHZ   :IN  STD_LOGIC;
CLK1MHZ    :OUT STD_LOGIC;
CLK100KHZ  :OUT STD_LOGIC;
CLK10KHZ   :OUT STD_LOGIC);
END COMPONENT;

COMPONENT CLKDIVIDE
PORT
(CLK50MHZ   :IN STD_LOGIC;
CLK25MHZ   :OUT  STD_LOGIC;
CLK100KHZ  :OUT STD_LOGIC;
CLK1MHZ    :OUT STD_LOGIC);
END COMPONENT;

COMPONENT VGA3
PORT
( CLK25,ENA_V            :IN STD_LOGIC;
REDOUT,GREENOUT,BLUEOUT  :OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
H_SYNC_OUT,V_SYNC_OUT,
BLANK,SYNC,WE_V,CLKVGA   :OUT STD_LOGIC;
DQ_V                :IN STD_LOGIC_VECTOR(15 DOWNTO 0);
ADR_V                                  :OUT STD_LOGIC_VECTOR (17 DOWNTO 0));
END COMPONENT;


COMPONENT DATINCTRL
PORT
(SRAM_DQ :INOUT   STD_LOGIC_VECTOR(15 DOWNTO 0);
DATIN       :IN   STD_LOGIC_VECTOR(7 DOWNTO 0);
OBF,CLK,ENA  :IN          STD_LOGIC;
ACKWE               :BUFFER  STD_LOGIC;
SRAM_ADR        :OUT     STD_LOGIC_VECTOR (17 DOWNTO 0);
OBF1,WE,IDLE :OUT     STD_LOGIC);
END COMPONENT;
```

“PAQUETE.vhd” Continuacion

```
COMPONENT CTRL_UNIT
PORT
(CLK25            : IN  STD_LOGIC;
 IDLE,PSH                 : IN  STD_LOGIC;
ENA_V,ENA_D           : OUT STD_LOGIC);
END COMPONENT;

COMPONENT BUSCTRL
PORT
(SRAM_ADR          :OUT STD_LOGIC_VECTOR(17 DOWNTO 0) ;
ADR_D,ADR_V       :IN STD_LOGIC_VECTOR(17 DOWNTO 0) ;
DQ_D                :IN STD_LOGIC_VECTOR(15 DOWNTO 0) ;
DQ_V                  :OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
SRAM_DQ          :INOUT STD_LOGIC_VECTOR(15 DOWNTO 0);
IDLE,WE_D,WE_V     :IN STD_LOGIC;
WE              :OUT STD_LOGIC;
CE,OE,UB,LB       :OUT STD_LOGIC);
END COMPONENT;

COMPONENT I2C_CNTR
PORT
(CLK, PSH, FIN : IN      STD_LOGIC;
ENA                   : OUT       STD_LOGIC;
REG_ADR,DATA_MSB,DATA_LSB      : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) );
END COMPONENT;

COMPONENT I2C_WR2A
PORT
(CLK,ENA              : IN      STD_LOGIC;
FIN,SLV_ENA,SCLO2   : OUT   STD_LOGIC;
SDA,SCL          : INOUT          STD_LOGIC;
REG_ADR,DATA_MSB,DATA_LSB : IN STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT;

COMPONENT I2C_SLAVE
PORT
(SLV_ENA,SCLO2       :IN    STD_LOGIC;
 SDA                 :INOUT STD_LOGIC);
END COMPONENT;

COMPONENT DINCMOS
PORT
(DQ_D                :INOUT   STD_LOGIC_VECTOR(15 DOWNTO 0);
 DATIN               :IN      STD_LOGIC_VECTOR(9 DOWNTO 0);
PXCLK,ENA_D,
LV,FV,CLK25          :IN   STD_LOGIC;
ADR_D               :OUT    STD_LOGIC_VECTOR (17 DOWNTO 0);
WE_D,IDLE,FVSIGO    :OUT    STD_LOGIC);
END COMPONENT;
```

“PAQUETE.vhd” Continuacion

```
COMPONENT BUSDESP
GENERIC (N : POSITIVE := 4; --Debe ser numero PAR de cols y rens debido al sensor
         M : POSITIVE := 2);
PORT
( PXCLK,CLK25,
FV,LV,VGA_ON          :IN STD_LOGIC;
FVSIGO,ENAVGA         :OUT STD_LOGIC;
CONTYO,CONTXO,
CONTYO2,CONTXO2       :OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
OUTWORD               :OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
INWORD                :IN  STD_LOGIC_VECTOR(9 DOWNTO 0));
END COMPONENT;

COMPONENT VGA4
PORT
( CLK25,ENA_V                 :IN STD_LOGIC;
REDOUT,GREENOUT,BLUEOUT  :OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
 H_SYNC_OUT,V_SYNC_OUT,
VIDEO_ON,
BLANK,SYNC,CLKVGA             :OUT STD_LOGIC;
DQ_V                          :IN STD_LOGIC_VECTOR(9 DOWNTO 0));
END COMPONENT;

COMPONENT BUFLINE
GENERIC
        (NBUF  : POSITIVE:=2;
        NWORD : POSITIVE:=10;
        NBITS   : POSITIVE:=10 );
 PORT
(  CLK_IN,CLK_OUT,VGA_ON  : IN    STD_LOGIC;
INWORD                : IN STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0);
REN_IN,REN_OUT        : IN INTEGER RANGE 0 TO NBUF-1;
COL_IN,COL_OUT        : IN  INTEGER RANGE 0 TO NWORD-1;
COL_OUTSIG            :OUT INTEGER RANGE 0 TO NWORD-1;
REN_OUTSIG            :OUT INTEGER RANGE 0 TO NBUF-1;
ENA_IN,ENA_OUT        : IN    STD_LOGIC;
OUTWORD               : OUT STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0));
END COMPONENT;

COMPONENT BUFDESP2
GENERIC (NWORD : POSITIVE :=10; --Debe ser numero PAR de cols y rens debido al sensor
         NBUF : POSITIVE  := 2);
PORT
( PXCLK,CLK25,
FV,LV,VGA_ON          :IN STD_LOGIC;
ENA_PX,
FVSIGO,ENAVGA         :OUT STD_LOGIC;
REN_IN,REN_OUT        :OUT INTEGER RANGE 0 TO NBUF-1;
CONT                  :OUT INTEGER RANGE 0 TO NWORD;
COL_IN,COL_OUT        :OUT INTEGER RANGE 0 TO NWORD-1 );
END COMPONENT;
```

“PAQUETE.vhd” Continuacion

```
COMPONENT BUFDESP3
GENERIC (NWORD : POSITIVE :=640 );
PORT ( PXCLK,CLK25,
FV,LV,VGA_ON          :IN STD_LOGIC;
START                 :IN STD_LOGIC;
ENA_PX,
ENAVGA                :OUT STD_LOGIC;
CONTWR                :OUT    INTEGER RANGE 0 TO 2*NWORD-1;
CONTRD                :OUT INTEGER RANGE 0 TO 2*NWORD-1 );
END COMPONENT;

COMPONENT BUFLINE2
GENERIC
        (NWORD   : POSITIVE:=640;
        NBITS   : POSITIVE:=10 );
PORT
(CLKWR                : IN    std_logic;
CLKRD                 : IN    std_logic;
INWORD                : IN    std_logic_vector(NBITS-1 downto 0);
WRADR                 : IN    INTEGER RANGE 0 TO 2*NWORD-1;
RDADR                 : IN    INTEGER RANGE 0 TO 2*NWORD-1;
WRENA,RDENA           : IN    std_logic;
OUTWORD               : OUT   std_logic_vector(NBITS-1 downto 0) );
END COMPONENT;

COMPONENT PSH
        PORT
        (CLK25          : IN  STD_LOGIC;
         PB             : IN  STD_LOGIC;
         START          : OUT STD_LOGIC);
END COMPONENT;

COMPONENT CONTPOS2
--      GENERIC (NWORD : POSITIVE :=640; --Debe ser numero PAR de cols y rens debido al sensor
--                       MLINE : POSITIVE :=480;
--                       NBITS  : POSITIVE :=10);

PORT ( CLK              :IN  STD_LOGIC;
    ENA                 :IN       STD_LOGIC;
    ENAD                :OUT STD_LOGIC;
    CONTYO,CONTCOL :OUT INTEGER RANGE 0 TO NWORD-1;
    CONTXO,CONTREN :OUT INTEGER RANGE 0 TO MLINE-1);
END COMPONENT;

COMPONENT FRAMEBEGIN
PORT
(CLK,START    :IN  STD_LOGIC;
 LV,FV        :IN       STD_LOGIC;
ENAFVLV,
FVSIGO        :OUT STD_LOGIC);
END COMPONENT;
```

“PAQUETE.vhd” Continuacion

```
COMPONENT PIPE
--GENERIC (NBITS : POSITIVE := 10;   --Ancho
--                  NTAP  : POSITIVE := 2 ;   --taps
--                  NWORD : POSITIVE := 640); --Longitud
PORT
(CLK : IN  STD_LOGIC;
ENA : IN  STD_LOGIC;
INP : IN  STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0);
TAP0,
TAP : OUT STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0);
OUTP: OUT STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0));
END COMPONENT;

COMPONENT PIPECONV2
--          GENERIC (NWORD  : POSITIVE :=640;--Debe ser numero PAR de cols y rens debido al sensor
--                              MLINE  : POSITIVE :=640;
--                              NBITS  : POSITIVE :=10;
--                              M                  : POSITIVE :=3 );
PORT
(CLK                          :IN  STD_LOGIC;
ENA                           :IN  STD_LOGIC;
CONTREN                       :IN  INTEGER RANGE 0 TO MLINE-1;
CONTCOL                       :IN  INTEGER RANGE 0 TO NWORD-1;
RED,GREN,BLUE                 :OUT STD_LOGIC_VECTOR (NBITS-1 DOWNTO 0);
INP,TAP,OUTP                  :IN  STD_LOGIC_VECTOR (NBITS-1 DOWNTO 0));
END COMPONENT;
COMPONENT BUFDESP4

PORT ( PXCLK,CLK25,
VGA_ON              :IN STD_LOGIC;
ENAD,ENAFVLV                        :IN STD_LOGIC;
WRENA,
ENAVGA              :OUT STD_LOGIC;
CONTWR                        :OUT INTEGER RANGE 0 TO 2*NWORD-1;
CONTRD                        :OUT INTEGER RANGE 0 TO 2*NWORD-1 );
END COMPONENT;

COMPONENT VGA5
PORT
( CLK25,ENA_V               :IN STD_LOGIC;
REDOUT,GREENOUT,BLUEOUT    :OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
H_SYNC_OUT,V_SYNC_OUT,
VIDEO_ON,
BLANK,SYNC,CLKVGA          :OUT STD_LOGIC;
REDIN,GREENIN,BLUEIN       :IN STD_LOGIC_VECTOR(9 DOWNTO 0));
END COMPONENT;
```

“PAQUETE.vhd” Continuacion

```
COMPONENT BUFRED
 PORT
(CLKWR                          : IN   std_logic;
CLKRD                             : IN   std_logic;
INRED                           : IN   std_logic_vector(NBITS-1 downto 0);
WRADR                           : IN   INTEGER RANGE 0 TO 2*NWORD-1;
RDADR                           : IN   INTEGER RANGE 0 TO 2*NWORD-1;
WRENA,RDENA                       : IN   std_logic;
OUTRED                            : OUT  std_logic_vector(NBITS-1 downto 0) );
END COMPONENT;

COMPONENT BUFGREEN
 PORT
(  CLKWR                        : IN   std_logic;
CLKRD                             : IN   std_logic;
INGREEN                         : IN   std_logic_vector(NBITS-1 downto 0);
WRADR                           : IN   INTEGER RANGE 0 TO 2*NWORD-1;
RDADR                           : IN   INTEGER RANGE 0 TO 2*NWORD-1;
WRENA,RDENA                       : IN   std_logic;
OUTGREEN                          : OUT  std_logic_vector(NBITS-1 downto 0) );
END COMPONENT;

COMPONENT BUFBLUE
 PORT
(CLKWR                          : IN   std_logic;
CLKRD                             : IN   std_logic;
INBLUE                          : IN   std_logic_vector(NBITS-1 downto 0);
WRADR                           : IN   INTEGER RANGE 0 TO 2*NWORD-1;
RDADR                           : IN   INTEGER RANGE 0 TO 2*NWORD-1;
WRENA,RDENA                       : IN   std_logic;
OUTBLUE                           : OUT  std_logic_vector(NBITS-1 downto 0) );
END COMPONENT;

COMPONENT PIPEBILINEAL
PORT
( PB,
PXCLK           :IN  STD_LOGIC;
FV,LV                       :IN       STD_LOGIC;
ENAD,ENAFVLV                :OUT STD_LOGIC;
INWORD                      :IN  STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0);
RED,GREEN,BLUE     :OUT STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0));
END COMPONENT;

COMPONENT DESP_REAL4
PORT
(ENAD,ENAFVLV,
PXCLK                   :IN     STD_LOGIC;
INRED,INGREEN,INBLUE    :IN     STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0);
REDOUT,GREENOUT,BLUEOUT     :OUT    STD_LOGIC_VECTOR(NBITS-1 DOWNTO 0);
H_SYNC_OUT,V_SYNC_OUT,
BLANK,SYNC,CLKVGA,WRENA,RDENA,ENAVGA         :OUT STD_LOGIC);
END COMPONENT;
```

“PAQUETE.vhd” Continuacion

```
COMPONENT MULTADD5
PORT
(R,G,B                          : IN    STD_LOGIC_VECTOR(9 DOWNTO 0);
RNO,GNO,BNO                     : OUT   STD_LOGIC_VECTOR(9 DOWNTO 0);
CLK,ENAD,ENAFVLV   : IN     STD_LOGIC;
ENAM                            : OUT   STD_LOGIC);
END COMPONENT;
END PAQUETE;
```

“CTES.vhd”

```
PACKAGE CTES IS

        CONSTANT NWORD:   INTEGER:=640;--número de píxeles en una línea
        CONSTANT MLINE:   INTEGER:=480;--número de líneas en el cuadro
        CONSTANT NTAP:    INTEGER:=2;  --números de buffer de línea para la tubería
        CONSTANT NBITS:   INTEGER:=10; --número de bits de un píxel
        CONSTANT M    :   NTEGER:=3;  --número de registros para la ventana

END CTES;
```