UNIVERSIDAD AMERICANA DE ACAPULCO
EXCELENCIA PARA EL DESARROLLO

FACULTAD DE INGENIERÍA EN COMPUTACIÓN

INCORPORADA A LA UNIVERSIDAD NACIONAL
AUTÓNOMA DE MÉXICO

UNA ARQUITECTURA FPGA PARA EL PROCESAMIENTO PIRAMIDAL DE IMÁGENES

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

PRESENTA:

MARCO AURELIO NUÑO MAGANDA

DIRECTOR DE TESIS

ING. GONZALO TRINIDAD GARRIDO

ACAPULCO, GRO. NOVIEMBRE DE 2002.

PAGINACION DESCONTINUA

Agradecimientos

A mi Madre:

Por haber puesto toda su confianza en mí, y porque gracias a ella nunca me ha faltado nada.

A la Universidad Americana de Acapulco:

Por haberme dado la oportunidad de ser alguien y porque en ella he vivido los mejores años de mi vida.

A la Facultad:

Por el apoyo incondicional recibido durante esos difíciles años.

Al mi asesor de tesis, el Ing. Gonzalo Trinidad Garrido:

Por haber estado siempre dispuesto a escuchar mis dificultades.

A los maestros:

Por haber dado el 100% dentro de las aulas y nunca haber buscado intereses personales.

Al Dr. Miguel Arias Estrada:

Por haberme proporcionado la oportunidad y ayuda necesaria para la elaboración de la presente tesis.

Índice General

Índice de Figuras

CAPITULO 2

CAPITULO 3

CAPITULO 4

CAPITULO 5

Introducción

Una imagen es una representación digital del mundo real que puede ser almacenada para posteriormente ser analizada y procesada por algún medio electrónico, como puede ser una computadora u otros medios. Se puede ver al vídeo como una secuencia de imágenes, en donde dependiendo de la fuente del vídeo, es el número de imágenes por segundo que se despliegan ante nuestras retinas. En el caso de la televisión, se despliegan 30 imágenes por segundo, mientras que en el cine se muestran 24 imágenes por segundo. Entonces, si se desea procesar un vídeo de 1 minuto de duración, tenemos que el número de cuadros necesarios para almacenar toda la información equivale a 1440 o 1800 cuadros dependiendo del sistema que se utilize, cantidad que representa mucha información aún con la potencia de cómputo disponible hoy en día.

Solo una pequeña parte de la información contenida en las imágenes puede ser relevante para una determinada tarea de visión. Esto quiere decir que es esencial eliminar o descartar información detallada en las etapas tempranas del análisis de imágenes. Es igualmente importante que el análisis sea llevado acabo con algoritmos que están estructurados de tal forma que sean altamente eficientes, y que puedan ser en ejecutados en sistemas hardware de alto rendimiento. Se puede ver a la obtención de la pirámide de una imagen como una forma de codificación de la imagen, en donde el principal objetivo es reducir tanto el tiempo de procesamiento como la complejidad de los algoritmos utilizados.

El procesamiento piramidal es una transformación que se le aplica a una imagen para obtener su pirámide. Una pirámide es un conjunto de imágenes obtenidas a partir de una imagen, cada una con una resolución menor a la original. El objetivo de obtener la pirámide de una imagen es trabajar con imágenes de diferentes resoluciones para minimizar el tiempo de procesamiento.

Organización de la tesis.

Este trabajo de investigación está organizado en los siguientes capítulos.

- Capítulo II. En este capítulo se exponen los conceptos relevantes que serán utilizados en la implementación del sistema. El concepto de mayor relevancia expuesto en este capítulo es el de pirámides y en especial la pirámide gausiana, que es una forma de filtrado de imágenes para posteriormente tomar determinados puntos y formar una imagen de menor resolución con una pérdida de información mínima.

- Capítulo III. En este capítulo se describen las herramientas utilizadas, tanto desde el punto de vista de hardware como desde el punto de vista de software. Se realizar una descripción detallada de los componentes de la tarjeta RC100 que son utilizados para la implementación, así como los programas utilizados para generar el archivo de configuración que es cargado en el FPGA (Field Programmable Gate Array). También se mencionan las bibliotecas de la tarjeta y la interacción de estos componentes con la arquitectura que se propone.

- Capítulo IV. En este capítulo se plantea la arquitectura propuesta, así como la descripción de sus módulos. Se puede descomponer a la arquitectura en tres grandes módulos: el módulo de obtención de imágenes, el módulo de procesamiento y el módulo de desplegado de resultados. Cada uno de estos módulos trata de utilizar la mayor cantidad de paralelismo posible, así como también se trata de tomar ventajas sobre las características específicas de la tarjeta RC100.

- Capítulo V. En este capítulo se muestran los resultados obtenidos tanto en la simulación funcional de alto nivel, como en la implementación en el FPGA. Se describe el proceso de prueba y se analizan detalladamente los resultados.

- Capítulo VI. En este capítulo se mencionan las conclusiones del trabajo, y también se proponen las tareas que pueden ser tomadas como trabajo futuro.

Capítulo 1

Planteamiento del problema y justificación

1.1 Planteamiento del problema

La idea de las pirámides es poder codificar la imagen utilizando una menor cantidad de espacio, para aplicarles algún procesamiento, y posteriormente regresar a la imagen original aplicando el procedimiento inverso, sin perder información valiosa. Lo que hace posible que se puede codificar la imagen en una menor cantidad de puntos es lo que llamamos correlación. Una correlación no es otra cosa mas que una medida de similitud entre dos conjuntos de datos que tienen el mismo número de elementos. En general, lo que pasa en una imagen es que los píxeles vecinos tienen un coeficiente de correlación muy alto, por lo que parte de la información en la imagen es redundante. Gracias a la técnicas de las pirámides, es posible obtener muestras que sean representativas de determinadas partes de la imagen, permitiendo esto minimizar el número de puntos necesarios para representar a la imagen

En la industria existen muchos procesos que requieren procesamiento de imágenes a gran velocidad. Dentro de la revisión previa realizada para esta investigación, sólo encontramos un sistema que utiliza el procesamiento piramidal como plataforma de procesamiento. Este sistema fue desarrollado por la empresa norteamericana Sarnoff, y entre las principales ventajas que ofrece es la integración de varias tareas en un mismo chip [Sarn1]. Su principal desventaja es su elevado costo, el cual supera por mucho al costo de un FPGA comercial. En lo referente a las aplicaciones de visión por computadora basadas en procesamiento piramidal, se encontraron muy pocos trabajos relacionados. Las aplicaciones relacionadas con el procesamiento piramidal son:

- Sistema de integración de mosaicos de imágenes [Bur2]
- Sistema de aprendizaje de texturas [Green1]
- Sistema de detección de movimiento [Brun1].

1.2 Justificación

Debido a que se trabajan con imágenes de menor tamaño, el tiempo de procesamiento de los algoritmos es mucho menor que si estos se aplicaran a la imagen original. Este es el motivo principal para utilizar procesamiento piramidal, y es el punto de partida para la implementación de una gran variedad de algoritmos de visión por computadora.

1.3 Objetivo General.

El objetivo de esta investigación es diseñar e implementar una arquitectura FPGA para el procesamiento piramidal en tiempo real, haciendo uso de técnicas de paralelismo, estructuras de pipeline, optimizaciones de accesos a memoria y registros de almacenamiento temporal. Para la implementación en el FPGA, se utilizará la tarjeta la tarjeta RC100 de Celoxica.

1.4 Objetivos Particulares

- Estudiar los conceptos básicos que serán de utilidad en el análisis y diseño de la arquitectura propuesta.

- Diseñar los módulos y efectuar la integración de los mismos, de tal forma que algoritmos de visión puedan ser implementados utilizando esta arquitectura.

- Probar y depurar la arquitectura propuesta, utilizando una tarjeta de prototipado específica.

1.5 Hipótesis.

- Los sistemas actuales de visión por computadora no utilizan el procesameinto piramidal como punto de partida, provocando que tarden un tiempo considerable en desempeñar la tarea de visión deseada.

- La implementación hardware de la arquitectura para el procesamiento piramidal además de desempeñar la obtención de pirámides en tiempo real, tendrá independencia de la computadora y un tamaño compacto, ya que se debe contar con una plataforma de procesamiento que pueda ser aplicado a aplicaciones científicas e industriales, como el control de robots o control de calidad, minimizando el tiempo de procesamiento de la aplicación.

Capítulo 2

Análisis Digital de Imágenes y Procesamiento Piramidal.

2.1 Representación digital de una imagen, muestreo y cuantificación

Una imagen que será procesada por computadora debe ser representada utilizando una estructura de datos discreta, como puede ser una matriz. Una imagen capturada por un sensor es expresada como una función f(x,y) continua de dos coordenadas en el plano [Son1]. La digitalización de imágenes significa que la función f(x,y) es muestreada en una matriz con M filas y N columnas. La cuantificación de la imagen asigna a cada muestra continua un valor entero (el rango continuo de la función f(x,y) se divide en K intervalos). Entre más fino sea el muestreo (esto es, entre más grandes sean los valores de N y M) y los niveles de cuantificación (K), mejor será la aproximación de la imagen almacenada [Te]1].

$$f(x,y) \approx \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,M-1) \\ f(1,0) & f(1,1) & \dots & f(1,M-1) \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ f(N-1,0) & f(N-1,1) & \dots & f(N-1,M-1) \end{bmatrix}$$

Figura 2-1: Representación matricial de una imagen

El muestreo de una imagen tiene como objetivo reducir la resolución espacial de la misma. Lo anterior quiere decir que entre menor sea el muestreo utilizado, menor será el espacio requerido para almacenar la imagen.

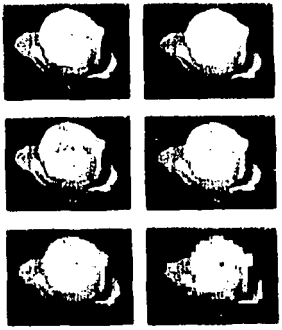

Figura 2-2: Efectos de aplicar diferentes tasas de muestreo a una misma imagen.

En la figura 2.2 se puede observar la perdida de información introducida con el aumento del paso del muestreo, así como el ruido que se va introduciendo en forma de patrones rectangulares sobre la imagen.

El efecto de la cuantificación viene dado por la imposibilidad de tener un rango infinito de valores de medida para la intensidad de brillo de los píxeles [Per1]. La tecnología actual permite llegar hasta los 16 bits de información en aplicaciones dedicadas, aunque por lo general se tienen 8 bits o 256 niveles de grises que son suficientes para que la vista humana aprecie la imagen con calidad fotográfica. En la figura 2.3 se muestra un conjunto de imágenes donde se muestra las diferencias entre el uso de diferentes niveles de cuantificación aplicados a una misma imagen.

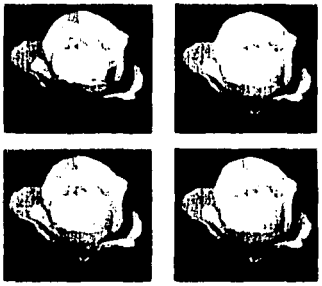

Figura 2-3: Efectos de aplicar diferentes tasas de cuantificación una misma imagen.

2.2 Filtrado

Los métodos de preprocesamiento local pueden ser divididos en dos grupos de acuerdo al objetivo del preprocesamiento. El primer grupo es el de suavizado, que tiene como objetivo suprimir ruido y pequeñas fluctuaciones en la imagen. Podemos decir que el suavizado es la operación equivalente a la supresión de altas frecuencias en el dominio de la transformada de Fourier. Desafortunadamente, el suavizado provoca pérdida en información de la imagen, como puede ser información relacionada con los bordes. El segundo grupo es el de operadores de gradiente que están basadas en derivadas locales de una imagen. Las derivadas son grandes en las localidades de la imagen donde la imagen cambia rápidamente [Son1].

Otra clasificación de los métodos de preprocesamiento se hace de acuerdo a las propiedades de la transformación. De acuerdo con esta clasificación puede haber transformaciones lineales y transformaciones no lineales [Del1]. Las transformaciones lineales calculan el valor resultante del píxel de salida como una combinación de brillo en una vecindad local O del píxel f(i,j) de la imagen de entrada.

2.3 Relaciones entre píxeles: vecindad y conectividad

Un pixel p de coordenadas (x, y) presenta un total de cuatro vecinos en el plano vertical y horizontal, siendo sus coordenadas:

	x , y – 1	
x - 1, y	x , y	x + 1, y
	x, y + 1	

Figura 2-4: Ejemplo de vecindad V_4

Este conjunto de píxeles se denomina vecindad de tipo 4 del píxel p, y se representa por N4(p) o V4(p) [Del1]. Además, se puede considerar la existencia de otros cuatro vecinos asociados a las diagonales cuyas coordenadas son:

x - 1, y – 1		x + 1, y – 1
	x, y	
x - 1, y + 1		x + 1, y + 1

Figura 2-5: Ejemplo de vecindad V_8

Los cuales se representa por ND(p). La suma de los anteriores define a los ocho vecinos del píxel p, N8(p).

Mediante el concepto de conectividad se quiere expresar que dos píxeles pertenecen al mismo objeto, por lo que esta relacionado con el de vecindad. Dos píxeles están conectados si son adyacentes (vecinos) y si sus niveles de gris satisfacen algún criterio de especificación (por ejemplo, ser iguales).

Existen tres tipos de conectividad:

1. Conectividad-4. Dos píxeles p y q presentan una conectividad-4 si q pertenece a N4(p).
2. Conectividad-8. Dos píxeles p y q presentan una conectividad-8 si q pertenece a N8(p)
3. Conectividad-m. Dos píxeles p y q presentan una conectividad-m si:

a. Q pertenece a N4(p), o
b. Q pertenece a ND(p) y el conjunto N4(p) ∩ N4(q) es el conjunto vacío.

2.4 Modelo RGB (Red-Green-Blue)

El color es una propiedad de enorme importancia para la percepción humana, pero no es tan utilizado en el procesamiento digital de imágenes, debido al costo computacional y a la memoria necesaria para procesar imágenes a color [Del1]. Algunas definiciones básicas para comprender los espacios de colores son:

- Brillo: Sensación que indica si un área está más o menos iluminada
- Tono: Sensación que indica si un área parece similar al rojo, amarillo, verde o azul, o a una proporción de los alguno de los colores.
- Coloración: Sensación por la que un área tiene un mayor o menor tono.
- Luminosidad: Brillo de una zona respecto a otra blanca de la imagen.
- Croma: La coloridad de un área respecto al brillo de un blanco de referencia.
- Saturación: La relación entre la coloridad y el brillo

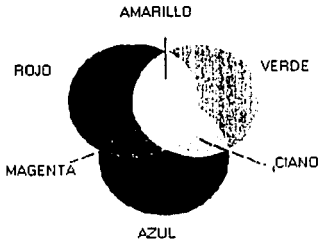

Figura 2-6: Modelo RGB

El color se relaciona con la propiedad de los objetos de reflejar ondas electromagnéticas de diferentes longitudes de onda [Ado1]. Los humanos detectan los colores como una combinación de los colores primarios (rojo, verde y azul).0

El Hardware generalmente despliega el color utilizado un modelo RGB (haciendo referencia a rojo, verde y azul). Cada uno de los píxeles está asociado con un vector tridimensional (r, g, b) el que se encarga de proveer las intensidades de color respectivas, donde (0, 0, 0) es negro, (k, k, k) es blanco, (k, 0, 0) es rojo "puro", en donde k = 255 y nos permite representar 256 niveles de cuantificación para cada color primario [Ado1].

2.5 Pirámides

Las pirámides son las estructuras de datos jerárquicas más simples. En análisis digital de imágenes, existen dos tipos, las M-pyramids (pirámides de matriz) y las T-pyramids (pirámides de árbol) [Son1]. Básicamente, el tipo de la pirámide se asocia la representación de los datos empleada por la estructura utilizada.

Una M-pyramid es una secuencia $\{M_L, M_{L-1}, \ldots M_0\}$ de imágenes, donde M_L tiene las mismas dimensiones y elementos que la imágenes original, y M_{L-1}, es derivada de la imagen M_L, reduciendo su resolución por un medio [Son1]. Cuando se crean pirámides, es ideal trabajar con matrices cuadradas que tengan dimensiones que sean potencia de 2, para que M_0 sea una imagen de un píxel (51).

Las M-pyramids son utilizadas cuando es necesario trabajar simultáneamente con imágenes a diferentes resoluciones. Una imagen que tiene una resolución menor contiene cuatro veces menos información, por lo tanto es procesada cuatro veces más rápido [Son1].

Una pirámide en sí es una forma de codificación de una imagen. Representar la imagen directamente en termino de los valores de gris de sus píxeles es algo muy ineficiente, dado que mucha de la información codificada es redundante [Bur1].

2.5.1 Pirámide Gausiana.

Inicialmente tenemos una imagen representada por el arreglo g_0 que contiene C columnas por R filas de píxeles. Cada píxel representa la intensidad en el correspondiente punto de la imagen y que ese píxel puede tomar el valor entre 0 y K. Esta imagen se considere el nivel 0 o el primer nivel de la pirámide gausiana. El primer nivel de la pirámide contiene a la imagen g_1 que es una versión reducida o filtrada de la imagen g_0. Cada valor dentro del nivel 1 es obtenido como un promedio ponderado de los valores en el nivel cero dentro de una ventana de 5x5. La imagen del nivel 2, representada por g_2, es obtenida a partir de los valores del nivel 1 y de la aplicación de la misma ventana de 5x5. Generalmente, la selección de la ventana de 5x5 se debe a que provee un filtrado adecuado con un bajo costo computacional [Bur1]. En la figura 2.7 se muestra el resultado de obtener los 5 niveles de la pirámide, partiendo de la imagen original o nivel 0.

Figura 2-7: Obtención de 5 niveles de la pirámide gausiana a partir de una imagen.

El proceso de promediado nivel a nivel es llevado a cabo por la función REDUCE.

g_k = REDUCE (g_k-1)

lo cual significa, que para los niveles 0<l<N y los nodos l,j,

$$g_l(i,k) = \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n) g_{l-1}(2i+m, 2j+n)$$

Donde N es el número de niveles de la pirámide, mientras que C_l y R_l son las dimensiones del $l$ésimo nivel. Nótese que la densidad de los nodos es reducido a la mitad por cada dimensión y a un cuarto en dos dimensiones de nivel a nivel.

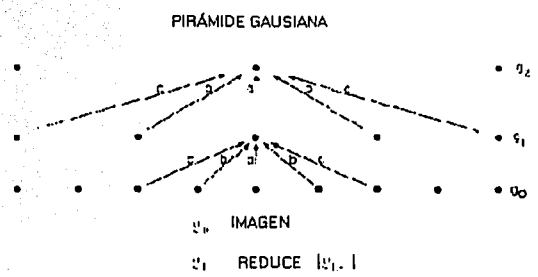

Figura 2-8: Obtención del siguiente nivel de la pirámide a partir del nivel anterior

Kernel o máscara utilizado.

La máscara o el Kernel utilizado es el mismo para generar cada nivel de la pirámide. Este patrón generalmente llamado el Kernel Generador, es seleccionado de acuerdo a ciertas características. En primer lugar, tenemos que hacer este separable [Bur1].

$$w(m,n) = \hat{w}(m)\hat{w}(n)$$

La función w es normalizada:

$$\sum_{m=-2}^{2} \hat{w}(m) = 1$$

y simétrica

$$\hat{w}(i) = \hat{w}(-i)$$

para i=0,1,2.

Entonces, para el siguiente vector de pesos:

0.15	0.20	0.30	0.20	0.15

Utilizando la metodología planteada anteriormente, la máscara a utilizar para el filtrado de los niveles de la pirámide es:

0.0225	0.03	0.045	0.03	0.0225
0.03	0.04	0.06	0.04	0.03
0.045	0.06	0.09	0.06	0.045
0.03	0.04	0.06	0.04	0.03
0.0225	0.03	0.045	0.03	0.0225

2.6 Sistemas de Tiempo Real.

Básicamente el concepto de sistema de tiempo real se refiere a un sistema que proporcione los resultados adecuados dentro de un determinado periodo de tiempo [Led1]. Lo anterior quiere decir que aunque un resultado sea correcto, carece de sentido si el tiempo de espera sobrepasa determinados límites.

Las áreas de aplicación de los sistemas de tiempo real son: robótica, multimedia, control de procesos industriales, sistemas de telecomunicaciones, sistemas espaciales, etc., esto es, todo aquel sistema donde la respuesta en tiempo es crítica para el buen funcionamiento. Generalmente se hablan de sistemas de tiempo real cuando se requieren de 10 a 50 respuestas por segundo (tratándose de video, se requieren 24 ó 30 respuestas por segundo).

2.7 Dispositivos Lógicos Programables (PLD)

Antes de la invención de los dispositivos lógicos programables, se utilizaban los dispositivos de lógica discreta fija, los cuales tiene las siguientes características [Xil1]:

- Un Chip contiene funciones lógicas fijas que no pueden ser reprogramadas.
- Las funciones de cada uno de los pines también son fijas

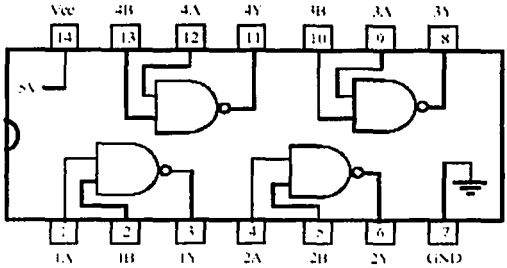

Figura 2-9: Esquemático del circuito 74LS00, que contiene 4 compuertas NAND de 2 entradas

Lo anterior significa que el usuario no puede cambiar la tarea llevada a cabo por el circuito, ni tampoco la función que desempeña cada pin del dispositivo.

El primer tipo de dispositivo programado por el usuario que podía implementar varios circuitos lógicos fue la memoria programable de sólo lectura (PROM), en donde las líneas de dirección son utilizadas como entradas y las líneas de datos como salidas. La PROM contiene un decodificador para sus líneas de dirección, por lo que las PROM son arquitecturas ineficientes para implementar circuitos lógicos, y raramente son utilizadas con este propósito [Xil1].

Las características de las PROM son:

- Se necesita realizar una decodificación completa de cada una de las direcciones utilizando arreglos de compuertas ANDs
- Los arreglos en fila de ORs definen salidas para cada dirección
- La salida de los pines es fija, pero el usuario puede reorganizar las entradas (líneas de dirección) y las salidas (datos).

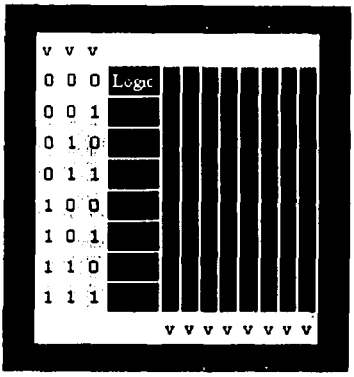

Figura 2-10: Estructura de una PROM. Las "v" superiores indican la entrada de datos (líneas de dirección) y las v inferiores indican la salida de datos.

El primer dispositivo desarrollado para implementar circuitos lógicos fue el Arreglo Lógico Programable (PLA). Un PLA consiste de dos niveles de compuertas lógicas: Un plano AND programable y un plano OR programable. Con esta estructura, un PLA es utilizado para implementar funciones lógicas en una forma de suma de productos. Estos dispositivos eran versátiles, dado que tanto los términos AND y OR pueden tener muchas entradas Los PLAs comenzaron a ser comercializados a inicios de los 70's por Philips. Sus principales desventajas fueron que eran muy caros y que ofrecían un desempeño bajo. Estas desventajas se atribuían a los dos niveles de lógica configurable, porque los planos lógicos programables eran difíciles de manejar e introducían retardos significativos [Xil1].

PLDs (Programming Logic Device)

Estos dispositivos también son conocidos como CPLDs. Las principales características de estos dispositivos son:

- Los arreglos lógicos de compuertas AND sólo decodifican la entrada especifica de los pines de interés. Los arreglos lógicos de compuertas ORs están, aunque están limitados a unos cuantos términos por celda lógica.

- La programación es relativamente simple, o de la misma complejidad que el lenguaje ensamblador.

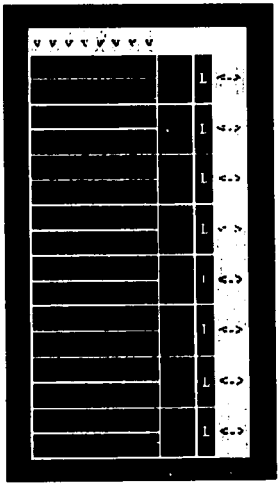

Figura 2-11: Estructura de un SPLD

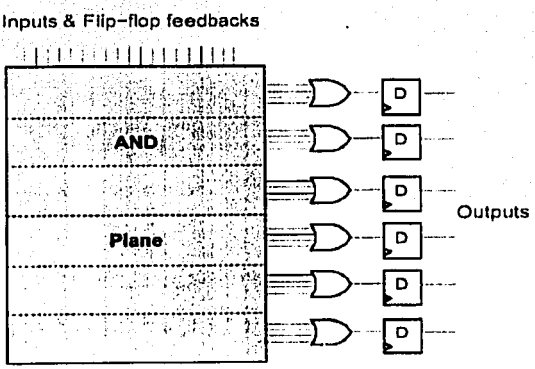

Figura 2-12: Estructura de un PLA

Complex Programmable Logic Device (CPLD)

Los CPLDs son dispositivos complejos que se componen de un arreglo de múltiples PLDs simples, de tal forma que éstos quedan organizados como bloques dentro del chip [Xil1]. Sus principales características son:

- Existe un grupo de celdas lógicas en la periferia de un dispositivo central de rutaje compartido.
- La programación es algo difícil dado que la interconexión entre los bloques lógicos no es una tarea trivial . Estos dispositivos de programas utilizando un lenguaje de descripción de hardware.

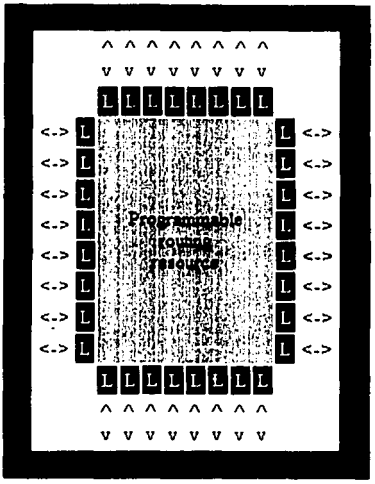

Figura 2-13: Estructura de una SPLD

FPGAs (Field Programmable Gate Array)

Los FPGAs ofrecen características flexibles para el diseño digital. Los FPGA's comerciales contienen flip-flops y circuitería combinacional. Cada fabricante de FPGAs incorpora diferentes características en los bloques lógicos y en los pads de entrada/salida de sus dispositivos, pero tienen en común un arreglo bidimensional de bloques lógicos rodeados en recursos de ruteo horizontales y verticales. Dependiendo de la familia de FPGAS, es el número de bloques lógicos, de pads de entrada y salida y de pines [Xil1].

Un bloque lógico de un FPGA puede ser programable para implementar una función lógica que tenga desde dos y hasta cuatro niveles combinacionales, y las salidas de cada bloque lógico pueden ser entradas en otros bloques lógicos [Brown1].

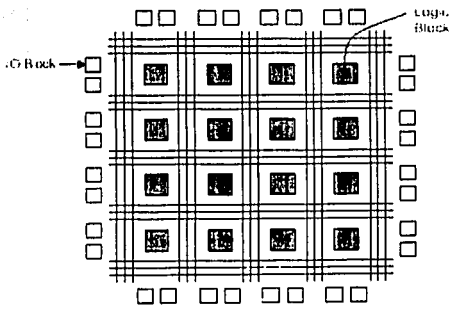

Figura 2-14: Estructura de un FPGA

- Esta arquitectura utiliza un arreglo bidimensional de celdas lógicas como si fueran islas en un mar de recursos de rutaje.

- El ruteo es mucho más complejo que en los CPLDs.

- Típicamente los FPGAs son programados en lenguajes como Verilog o VHDL. En este trabajo de investigación, se utilizó un lenguaje de programación diseñado por Celoxica, llamado Handel-C.

- Comparando el nivel de complejidad, podríamos decir que es similar a la programación en Lenguaje C.

2.7 VHDL.

La siglas VHDL significan VHSIC Hardware Description Languange. VHSIC es una abreviación de Very High Speed Integrated Circuit (circuito integrado de muy alta velocidad). En sí VHDL es un lenguaje que puede describir el comportamiento y la

estructura de sistemas electrónicos, pero especialmente es utilizado como un lenguaje para diseños de hardware digital como son los ASICs, FPGAs y de algunos circuitos MSI.

2.8 Handel-C.

La compilación hardware es una herramienta poderosa para el codiseño hardware/software. Convencionalmente es común diseñar un sistema utilizando un lenguaje de programación de alto nivel, como C, y entonces diseñar la parte de hardware utilizando un lenguaje de descripción de hardware, como VHDL. La compilación hardware utilizando Handel-C simplifica el proceso de diseño, dado que provee la característica de utilizar un lenguaje C de alto nivel para generar hardware [Pet1].

Handel-C es un lenguaje de programación diseñado para compilar programas en imágenes de hardware de FPGAs y ASICs. Básicamente es un subconjunto de C, junto con algunos constructores para configurar el dispositivo hardware y para soportar la generación de hardware eficiente. Provee de las expresiones necesarias para describir algoritmos complejos, pero también permite características de alto nivel, como son los apuntadores y la aritmética de punto flotante. Los programas son mapeados en hardware a nivel de una lista de componentes, y este mapeo puede estar tanto en formato Xnf o en formato Edif.

El enfoque de Handel-C es muy diferente al enfoque de VHDL. VHDL es un lenguaje que permite la creación de circuitos sofisticados. Por lo general se espera que el diseñador tenga conocimientos de bajo nivel relacionados con el hardware, y por lo general lo pone a pensar en términos de compuertas o flip-flops [Pet1]. En contraste con la anterior, Handel-C es una herramienta enfocada a la optimización a nivel algoritmo, dejando a un lado los detalles de diseño y permitiendo la creación de hardware altamente especializado.

Cuando se utiliza C como el lenguaje de descripción, la longitud más pequeña para un entero es de ocho bits, lo cual significa que desperdicia siete registros cuando este entero es utilizado como una bandera. Handel-C provee operaciones de manipulación de bits y la posibilidad de paralelizar procesos tanto de operaciones simples

como de módulos completos. El paralelismo no es posible cuando se utilizan lenguajes secuenciales.

En Handel-C, todas las decisiones a nivel compuerta y de optimización del diseño son hechas por el compilador de tal forma que el programador se concentre sólo en la tarea que desea implementar [Pet1]. Como consecuencia de lo anterior, el tiempo de diseño en Handel-C es mucho menor comparado con el tiempo de diseño utilizando lenguajes de descripción de hardware, además de que es una herramienta utilizada para programadores que no tienen mucho conocimiento de hardware.

Las principales características de Handel-C se resumen a continuación:

- Handel-C es un lenguaje de alto nivel que requiere que se piense en términos de algoritmos en vez de pensar en términos de circuitos
- Handel-C utiliza un modelo de retardo cero y un estilo de diseño síncrono
- Handel-C es implícitamente secuencial. Los procesos paralelos deben ser especificados
- Todo el código en Handel-C puede ser sintetizado.
- Las señales en Handel-C son diferentes a las señales en VHDL; son asignadas inmediatamente y mantiene su valor por un ciclo de reloj
- Handel-C tiene conceptos abstractos de alto nivel, como son los apuntadores.

Capítulo 3

Descripción de las herramientas utilizadas

3.1 Herramientas de software utilizadas para la implementación.

En esta sección se describen los programas de software utilizados para el diseño e implementación de la arquitectura propuesta.

3.1.1 DK1 Design Suite.

DK1 Design Suite es una herramienta que soporta el diseño, validación, refinamiento iterativo e implementación de algoritmos complejos en hardware. Incluye módulos de simulación y síntesis, los cuales son manejados por Handel-C. Handel-C es un lenguaje basado en el estándar ANSI-C, con conceptos como temporización, concurrencia, tamaños flexibles de variables y manejo de recursos que permite a los ingenieros de software y hardware implementar rápidamente algoritmos completos de manera eficiente en hardware [Cel3].

DK1 Design Suite fue utilizado para la descripción de alto nivel de nuestra arquitectura. Al finalizar el proceso de síntesis, lo que obtenemos es un archivo EDIF.

Las siglas EDIF significan Formato de Intercambio de Diseño Electrónico. Este formato es utilizado para intercambiar datos entre diferentes sistemas CAD y entre los sistemas CAD y la fabricación y ensamble de circuitos impresos. La palabra electrónica se refiere al tipo de datos utilizados y no al mecanismo de intercambio. Un archivo EDIF puede ser leído y puede ser intercambiado electrónicamente. A los sistemas CAD para el diseño de circuitos electrónicos también se les conoce como Sistemas de Diseño Electrónico Asistido por Computadora (ECAD) o Sistemas de Automatización de Diseño

Electrónico (EDA). La sintaxis de los archivos EDIF está basada en el lenguaje de programación LISP [Eda1].

3.1.2 Foundation ISE (Integrated Synthesis Environment)

Foundation ISE es una herramienta que se utilizada para generar el archivo de programación del FPGA a partir del archivo EDIF generado por el DK1 Design Suite. Esta herramienta permite administrar proyectos, trabajar con editores y simuladores para definir las especificaciones particulares del proyecto. Se dispone de una serie de herramientas de implementación para compilar y redefinir aspectos de los diseños.

Hay varias formar de representar un diseño. A estas representaciones se les denominan fuentes. Estas fuentes incluyen no sólo la descripción de los circuitos, como son diagramas esquemáticos o código de descripción de hardware, sino también restricciones y documentación del diseño.

En el caso de nuestra implementación, el tipo de archivo fuente utilizado es el EDIF. Como se definió en la sección anterior, los archivos EDIF son archivos de intercambio entre Sistemas de Diseño Electrónico Asistido Por Computadora. Lo que este programa hace con el archivo EDIF. proporcionado es obtener el mapa de conexiones internas para el dispositivo FPGA que efectué la tarea que se desea implementar con la arquitectura diseñada.

3.1.3 FTU (File Transfer Utility).

FTU es una herramienta que es utilizada para transferir el archivo de programación a la memoria FLASH del FPGA. Posteriormente, se conectan los dispositivos necesarios a la tarjeta y se verifica el funcionamiento de la aplicación.

Las características de la aplicación FTU son:

- Funciona en cualquier versión de 32 bits de Windows (95/98/Me/NT4/2000)
- Transfiere archivos .bit a un FPGA

- Transfiere archivos o datos en formato RAW a una localidad de la Memoria Flash especificada por el usuario.
- Tiene soportes para scripts (Esto es, pueden escribirse programas por lotes)
- Para la transferencia de archivos de programación al FPGA, tarda aproximadamente 5 segundos por MB de información.
- Para la transferencia de datos a la Memoria RAM, tarda aproximadamente 60 segundos por MB de información.

3.2 Descripción de los dispositivos utilizados para la implementación.

En esta sección se describen los dispositivos utilizados para el diseño e implementación de la arquitectura propuesta.

3.2.1 Computadora

Las características de la computadora utilizada son:

- Procesador Pentium III a 866 Mhz
- Disco Duro de 20 Gb
- Memoria RAM de 256 MB
- Teclado y Ratón
- Monitor de 15 Pulgadas.

En la computadora fueron realizadas las siguientes actividades:

a) El diseño de la arquitectura utilizando la herramienta DK1 Design Suite, para posteriormente realizar la síntesis del diseño (generación del archivo EDIF).

b) La generación del archivo de configuración para la tarjeta RC100 (archivo .BIT), utilizando la herramienta Foudation ISE de Xilinx.

c) La transferencia del archivo .BIT a la tarjeta RC100, a través del puerto paralelo y utilizando la utilería FTU de Celoxica.

Además, algunos periféricos (como son el Monitor, el ratón y el teclado) fueron utilizados para conectarse a la tarjeta y verificar el funcionamiento de las arquitecturas implementas. El monitor fue utilizado principalmente para visualizar los resultados arrojados por la versión final de la arquitectura propuesta en esta tesis.

3.2.2 Cámara de vídeo

Podemos utilizar cualquier tipo de cámara que nos genere una salida de Super Vídeo o en formato PAL o NTSC. En este caso se utilizó una cámara Handicam fabricada por Sony, y específicamente el modelo CCD-TR818.

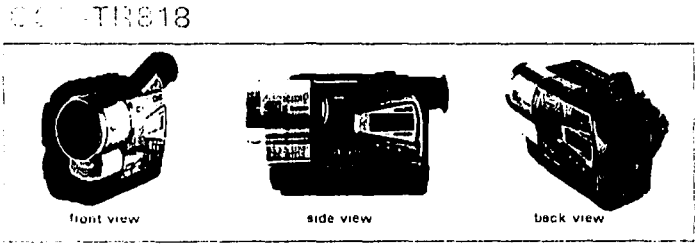

Figura 3-1: Cámara utilizada para la captura de Imágenes.

3.2.3 Tarjeta RC100 de Celoxica.

La tarjera RC100 de Celoxica es una tarjeta didáctica que tiene como objetivo introducir a los estudiantes al uso de la herramienta Handel-C y a la programación de los FPGAs [Cel1].

Esta tarjeta contiene dos dispositivos programables:

- XILINX Spartan II FPGA. Es un FPGA con capacidad de 200,000 compuertas equivalentes.

- XILINX XCR3128XLD CPLD. Es utilizado para manejar la configuración del FPGA utilizando el puerto paralelo o la memoria FLASH de la tarjeta.

Los conectores principales de esta tarjeta son:

- Interruptor de encendido/apagado. Básicamente sirve para encender y apagar la tarjeta.

- Fuente de poder. Sirve para alimentar a la tarjeta.

- Conector Paralelo. Tiene la función de transferir archivos de la computadora a la tarjeta y viceversa.

- Interruptores DIP. Esta tarjeta trae algunas aplicaciones ya cargadas. Mediante estos interruptores podemos cargarlas para su ejecución en el momento de encender la tarjeta.

- Conector de ratón y de teclado. Podemos incluir en nuestras aplicaciones el uso del ratón y del teclado para introducir datos.

- Salida de vídeo. Una vez realizado un proceso, la tarjeta nos puede mostrar el resultado de las operaciones

- Entradas de vídeo S y CSB. Esta tarjeta puede recibir como entrada vídeo en los formatos NTSC y PAL.

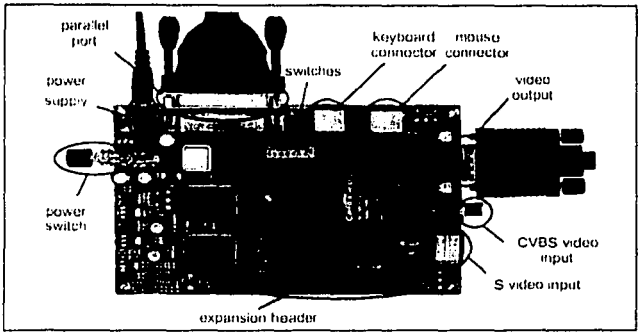

Figura 3-2: Principales conectores e interruptores de la tarjeta RC100

3.2.4 Bibliotecas utilizadas

Existen bibliotecas especialmente diseñadas para la tarjeta RC100 que pueden ser utilizadas para la implementación de determinados diseños [Cel2].

Memoria.

- Controlador para la Memoria RAM. Este controlador nos permite leer y escribir a la memoria Flash en los modos de 8 y de 16 bits
- Macros para habilitar la reconfiguración del FPGA a partir de una determinada localidad de la memoria Flash
- Macros para acceder a la memora SSRAM (Synchonous Static Random Acess Memory). Este nos permite leer y escribir a los dos bancos RAM tanto por byte como por palabra.

Vídeo

- Generador de sincronía y un Controlador para el DAC. Esto facilita la generación de vídeo VGA en monitores estándar
- Controlador para el Decodificador de vídeo SAA7111. Esto nos permite obtener vídeo tanto en formato PAL como NTSC.

Puerto paralelo

- Macros para habilitar la comunicación PC-Tarjera RC100 a través del puerto paralelo

3.3 Componentes principales

En esta sección se describen los dispositivos utilizados para el diseño e implementación de la arquitectura propuesta.

3.3.1 Interruptores DIP

Los interruptores DIP en la tarjeta permiten al usuario establecer la localidad de la memoria Flash a partir de la cual la tarjeta RC100 cargue en el momento en el que la tarjeta es encendida. Para utilizar las aplicaciones preinstaladas es necesario botear a partir del sector 0. Los interruptores están organizados de igual forma que un contador binario [Cel2]. Por ejemplo, para cargar del sector 1, es necesario encender el primero y el último de los interruptores (bit 1 y el autobut). Para cargar del sector 6, es necesario encender el segundo y el tercer interruptor, además del interruptor de autoboot.

En el momento de realizar la transferencia del archivo de programación del FPGA por el puerto paralelo, el autoboot debe estar en posición de OFF y el indicador del sector en donde será cargada la aplicación definida por el usuario debe de estar en cero.

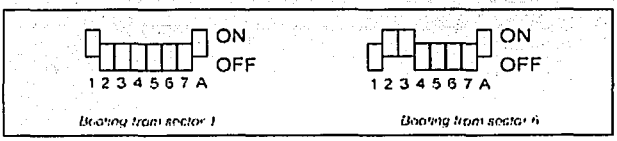

Figura 3-3: Ejemplo de las posiciones del los interruptores DIP para la carga de aplicaciones localizadas en diferentes sectores de la memoria Flash.

3.3.2 CPLD

La tarjeta RC100 tiene un CPLD, el cual es utilizado para configurar el FPGA a partir de varias fuentes de datos [Cel2]. El CPLD tiene acceso a:

- El puerto paralelo
- La memoria FLASH
- El FPGA

El CPLD está conectado a la memoria Flash, la cual se divide en 64 localidades booteables. Los interruptores DIP indicarán al FPGA de que sector de arranque cargar en el momento del encendido de la tarjeta.

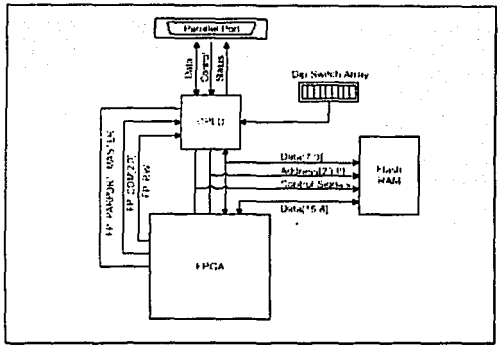

Figura 3-4: Principales líneas de conexión entre el CPLD, el FPGA y el conector del puerto paralelo.

Dependiendo de switch de Autoboot que esté habilitado, el CPLD configura al FPGA a partir de la memoria Flash cuando se enciende la tarjeta. El archivo de configuración es leído a partir de la memoria Flash, el cual es especificado por el sector de arranque seleccionado por los interruptores DIP. De hecho, el FPGA puede ser configurado a través del CPLD con información almacenada en alguna localidad de la memoria Flash.

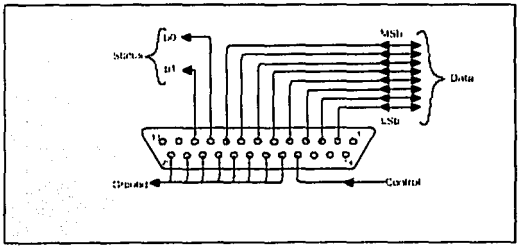

Figura 3-5: Pines del conector paralelo utilizados para la transferencia de datos entre la Tarjeta RC100 y la computadora

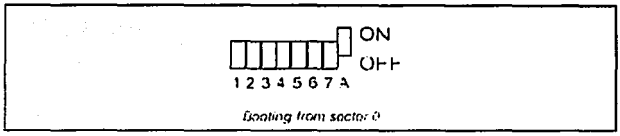

Figura 3-6: Configuración de los Interruptores DIP para cargar la aplicación localizada en el sector 0 de la Memoria Flash.

3.3.3 Flash RAM.

La tarjeta RC100 tiene una dispositivo de memoria flash de 60 Mbits. Esta memoria se encuentra dividida en 64 bloques, cada uno con 128KB. Todas las señales requeridas para operar el dispositivo en los dos modos posibles de operación (8 Bits y 16 Bits), están mapeadas directamente al FPGA [Cel2].

El proceso de borrado de un bloque consiste en poner todos los bits en el bloque a uno. Un bloque debe ser borrado antes de ser programado. Esto es porque durante la programación, los unos pueden ser convertidos a ceros pero no de forma inversa.

3.3.4 SSRAM

La tarjeta contiene dos bancos de memoria de 36-bits por 256K. Estas memorias son capaces de operar a velocidades de más de 100 Mhz. Todos los pines están directamente conectados con el FPGA [Cel2].

Cada una de las localidades de 36 bits de la memoria está organizada en 4 bytes y en 4 bits de paridad (uno para cada byte)

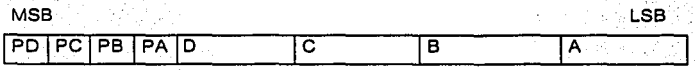

Figura 3-7: Organización de la palabra de 36 bits de la SSRAM

Se proveen macros separadas para acceder a cada banco de SSRAM, así como macros para leer los bytes individuales y sus respectivos bits de paridad.

3.3.5 FPGA.

La tarjeta RC100 contiene un FPGA Xilinx Spartan II con una capacidad de 200,000 compuertas equivalentes. Este dispositivo es la parte principal de la tarjeta y es la pieza principal de lógica reconfigurable que los usuarios pueden utilizar [Cel1].. Este FPGA tiene conexiones directas con los siguientes dispositivos:

- Dos bancos de memoria SSRAM.
- Una memoria Flash.
- Un convertidor Analógico-Digital DAC.
- Decodificador de vídeo
- Conectores PS/2 para teclado y para ratón
- Leds
- Dos displays de 7 segmentos
- Un conector de expansión.

El FPGA también accede al puerto paralelo a través del CPLD. El acceso se permite a todos los pines de datos y también a ciertos pines de control y estatus.

3.3.6 Decodificador de vídeo NTSC/PAL

La tarjeta contiene un decodificador de vídeo SAA7111 de Philips, que le permite al FPGA capturar y decodificar vídeo en formato NTSC y PAL. El vídeo decodificado

puede ser puesto en un formato de 16 o 24 bits RGB o en formato YCrCb. Todas las señales de control a partir de este chip están mapeadas directamente a los pines del FPGA [Cel2]. El chip decodificador es controlado utilizando un bus.

El decodificador SAA7111 puede decodificar señales de PAL y NTSC de varios formatos y convertirlas en una cadena de valores de 16 bits digitalizados. El chip acepta entradas de CVBS compuesto o de una fuente de video.

El controlador de video para el SAA7111 funciona a través de dos dominios de reloj. Un dominio es manejado por el reloj generado por el chip, a 27 Mhz. El otro es el dominio de reloj principal, donde se utilizan los datos generados por el chip. Se utiliza un buffer FIFO para pasar datos entre los dominios de reloj.

3.3.7 Subsistema de salida de vídeo.

La tarjeta RC100 tiene la propiedad de generar salida VGA de 24 Bits para ser desplegada en un monitor o en un proyector. La tarjeta contiene un Convertidor Analógico Digital (DAC) que convierte la salida digital de video del FPGA a las señales digitales requeridas por el conector VGA [Cel1]. El DAC también alimenta a las señales Sync y Blanking por el hardware generador de sincronía para permitir poner todos los voltajes a los niveles de blanqueo apropiados (como se define en el estándar VGA) durante el periodo de blanqueo.

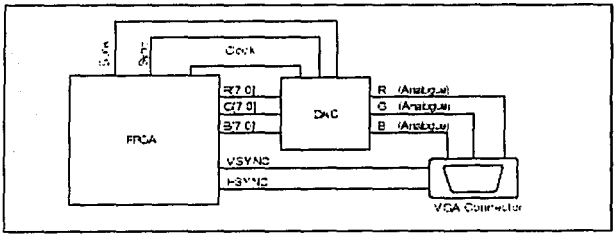

Figura 3-8: Principales conexiones el DAC y el FPGA

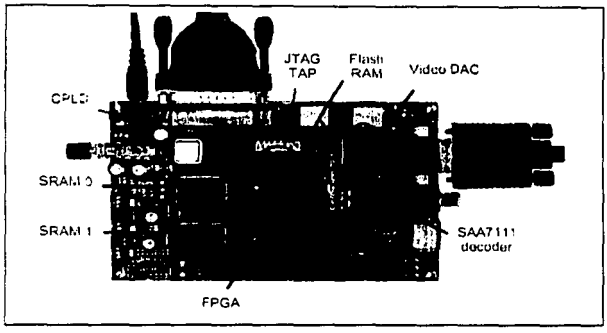

Figura 3-9: Conectores principales de la tarjeta RC100

3.3.8 Generador de Salida VGA.

Para generar salida VGA, la librería RC100 contiene los siguiente elementos:

- Un macro de control de video (RC100VideoDriver)
- Macros para obtener información relacionada con la pantalla
- Funciones para manipular los valores de la salida del decodificador.

RC100VideoDriver contiene un generador de sincronía que genera pulsos horizontales y verticales de sincronía basados en el estándar VGA640x480 [Cel1].

El controlador proporciona como salida la posición X y Y del barrido en la pantalla. Se puede asignar el valor a la pantalla en una esquema de píxel por píxel. Este valor debe ser asignado en cada ciclo de reloj. En otras palabras la velocidad del reloj del diseño es igual a la tasa de reloj del píxel. Lo anterior quiere decir que para desplegar una imagen de 640 x 480, necesitamos 640 x 480 pulsos de reloj.

Existe una estructura de datos que encapsula todas las variables asociadas con el controlador [Cel2], la cual se muestra a continuación:

```
typedef struct
{
signal unsigned 24 Output
signal unsigned 10 ScanX
signal unsigned 10 ScanY
signal unsigned 1 Visible
signal unsigned 1 HBlank
signal unsigned 1 VBlank
} RC100_VGA_DRIVER;
```

Figura 3-10: Estructura utilizada por el controlador de vídeo para obtener datos del DAC

Miembros	Descripción
Output	Señal que toma el valor RGB de 24 bits que será desplegado
ScanX	Señal indicando el píxel horizontal actual que esta siendo desplegado
ScanY	Señal indicando el píxel vertical actual que está siendo desplegado

Tabla 3.6: Descripción de los miembros más importantes de la estructura.

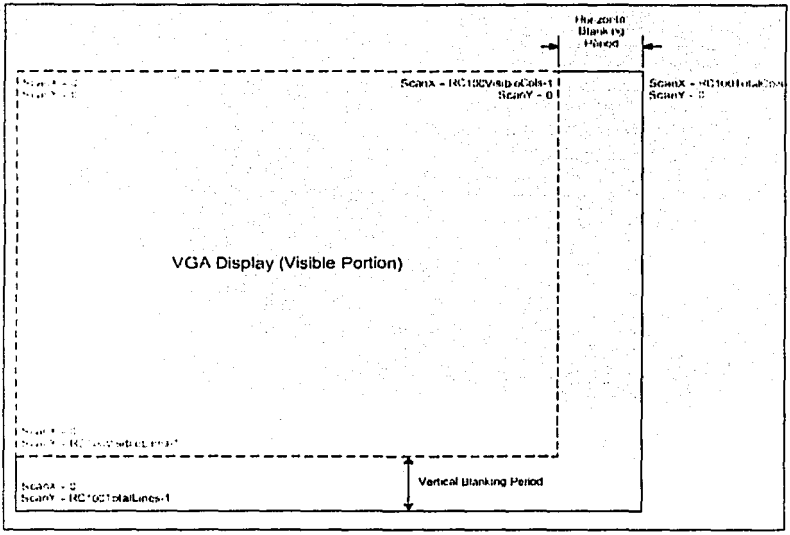

Figura 3-11: Periodos de visualización y de blanqueo de imágenes en la salida VGA

Capítulo 4

Diseño de la Arquitectura

4.1. Representación matricial vs representación vectorial de una imagen

En general, para representar una imagen, los programas de procesamiento digital de imágenes (como son Matlab o Halcón) utilizan matrices. Dependiendo del tipo de imagen es el tipo de matriz utilizada. Cuando se trata de una imagen en escala de grises, se utiliza una matriz de tamaño N x M, donde N es el número de filas y M es el número de píxeles por fila de la imagen. Cuando se trata de una imagen codificada a colores utilizando un modelo RGB, entonces se puede ver a la imagen como un conjunto de matrices, que en el caso del modelo RGB son tres, una para cada componente de la imagen, y donde cada matriz es de tamaño N por M.

A diferencia de los programas de procesamiento digital de imágenes, en el diseño de una arquitectura FPGA para el procesamiento de imágenes, vemos a la imagen desde una perspectiva diferente. Dado que no existen memorias matriciales se deben utilizar memorias vectoriales. Por consiguiente, para efectos del procesamiento de una imagen, ya no se ve a la imagen como una matriz, sino como un vector.

Se parte de la suposición inicial de que la imagen está representada de la siguiente manera: Una fila de una matriz representa una fila de la imagen, pero en una memoria ya no observamos una fila. Se supone que se utilizan K bits para representar los datos de una imagen. En notación matricial, se ve una fila que contiene N elementos de tamaño K. El problema con las memorias es que no se puede tener palabras de tamaño K*N, sino que se tiene palabras de menor tamaño. Sea J el tamaño de la palabra (en bits)

de la memoria que será utilizada para almacenar la imagen. Se debe buscar una forma de representar esa fila en una columna. Entonces, si J=K, para almacenar una sola fila de la imagen, se necesitan K*N posiciones de memoria, dado que se almacena un pixel por palabra. Si J = K*2, se necesitan K*N/2 posiciones de memoria, dado que se almacenan dos pixeles por palabra.

Entonces, para almacenar toda la imagen en una memoria RAM, se guarda primero una fila de la imagen. Cuando se termina de almacenar una fila, se almacena la siguiente a partir de la siguiente posición de memoria disponible. Para almacenar una imagen de N*M pixeles en memoria, necesitamos N*M posiciones de memoria. Entonces, para almacenar una imagen de 480*512, necesitamos 245,760 posiciones de memoria.

Para ser consistente con el tipo de numeración utilizada en sistemas digitales, se comienza a contar a partir de 0 y no a partir de 1 como se hace en notación matricial. Para este fin, el pixel (1,1) en notación matricial es el pixel (0,0) en notación vectorial, el pixel (1,2) en notación matricial es el pixel (0,1) en notación vectorial, y así sucesivamente.

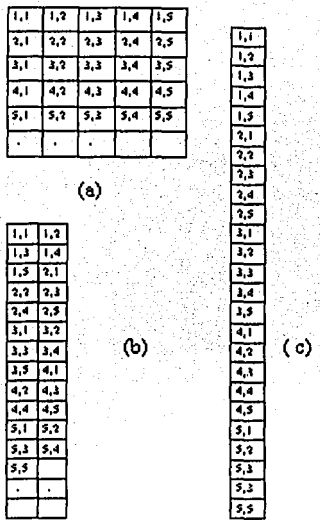

Figura 4-1: (A) representación matricial de una imagen (b) Representación vectorial cuando el tamaño de palabra de la memoria es del mismo tamaño que el espacio requerido para almacenar un píxel (c) Representación vectorial cuando el tamaño de palabra de la memoria es el doble del espacio requerido para almacenar un píxel

Ahora, para acceder a cualquier píxel en nuestra memoria RAM, se necesita realizar el cálculo de la dirección. En caso de que el tamaño de palabra de la memoria sea igual al espacio requerido por un píxel de nuestra imagen, la fórmula a utilizar se muestra a continuación:

$$Direccion(i, j) = i * Numero\_de\_filas + j$$

Así por ejemplo, si tenemos una imagen de 5x5 y queremos acceder al píxel (0,0) aplicando la formula obtenemos:

$$Direccion(i,j) = i * Numero\_de\_filas + j$$
$$Direccion(0,0) = 0 * 5 + 0 = 0$$

Si queremos acceder al píxel (2,3) obtenemos:

$$Direccion(i,j) = i * Numero\_de\_filas + j$$
$$Direccion(2,3) = 2 * 5 + 3 = 13$$

Si queremos acceder al píxel (4,2) obtenemos

$$Direccion(i,j) = i * Numero\_de\_filas + j$$
$$Direccion(4,2) = 4 * 5 + 2 = 22$$

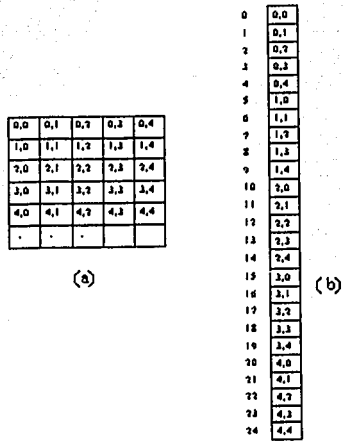

Figura 4-2: Organización de una imagen en una memoria de longitud de palabra igual al tamaño de bits necesario para almacenar un píxel.

En caso de que el tamaño de la palabra de la memoria RAM fuera el doble del espacio que requerimos para almacenar un pixel, se pueden almacenar dos píxeles por palabra. Entonces, la fórmula para el cálculo de la dirección es:

$$Direccion(i, j) = i * (Numero\_de\_filas / 2) + j \setminus\setminus 1$$

Donde "\\1" representa un corrimiento hacia la derecha en Handel-C. Dado que hay dos píxeles almacenados en la misma dirección de memoria, el operador de corrimiento se aplica para acceder a la misma dirección en caso de que se quiera obtener cualquiera de los dos píxeles. Esta operación tiene sus ventajas y desventajas que a continuación enumero:

Ventajas

a) Se puede acceder a dos píxeles en el mismo ciclo, por lo que el número de lecturas a memoria se reduciría en un factor de un ½.
b) En caso de requerir un procesamiento, este se haría en un tiempo mucho menor.

Desventajas

a) En caso de que se desee acceder a dos píxeles, sino que sólo un pixel, daría el mismo caso tener una memoria que almacene un pixel en vez de dos, dado que se harían el mismo número de accesos a memoria.

b) Se necesitaría un bus de datos mucho más grande, dado que al leer de memoria leeríamos dos píxeles en vez de uno, por lo que la implementación de nuestra arquitectura necesitaría más recursos hardware.

A continuación, se muestra un esquema de acceso a píxeles.

Fila	Columna	Codificación binaria de la columna	Codificación binaria de la columna con corrimiento	Equivalente decimal de la columna	Resultado del cálculo de la dirección
0	0	0000	000	0	0
0	1	0001	000	0	0
0	2	0010	001	1	1
0	3	0011	001	1	1
0	4	0100	010	2	2
0	5	0101	010	2	2

Tabla 4.1: Calculo de direcciones en una memoria donde la palabra puede almacenar dos bits.

Así por ejemplo, si tenemos una imagen de 5x4 y queremos acceder al píxel (3,0) aplicando la fórmula obtenemos:

$$Direccion(i, j) = i * Numero\_de\_filas + j \setminus\setminus$$
$$Direccion(3,0) = 3 * 2 + 0 \setminus\setminus = 6 + 0 = 6$$

Si quisiéramos acceder al píxel (4,1) obtenemos:

$$Direccion(i, j) = i * Numero\_de\_filas + j \setminus\setminus$$
$$Direccion(4,1) = 4 * 2 + 1 \setminus\setminus = 8 + 0 = 8$$

Entonces, para un par de píxeles consecutivos, se accede a la misma dirección de memoria. Para determinar que parte de la localidad leída se necesita, basta con comprobar si el píxel al que se desea acceder es par o impar. Si es un píxel par, tomaremos la primera parte de la localidad (en este caso, los primeros 16 bits). En caso de ser un píxel impar, tomaremos la segunda parte de la localidad (en este caso, los segundos 16 bits).

Siendo de esta forma, entonces para el acceso a los píxeles anteriores:

- Para el píxel (3,0), tomaríamos la primera parte de la localidad.
- Para el píxel (4,1), tomaríamos la segunda parte de la localidad.

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3
4,0	4,1	4,2	4,3
5,0	5,1	5,2	5,3

(a)

0	0,0	0,1
1	0,2	0,3
2	1,0	1,1
3	1,2	1,3
4	2,0	2,1
5	2,2	2,3
6	3,0	3,1
7	3,2	3,3
8	4,0	4,1
9	4,2	4,3
10	5,0	5,1
11	5,2	5,3
12	...	...
	...	...
	.	.

(b)

Figura 4-3: Representación del direccionamiento en una memoria que contiene dos píxeles por localidad de memoria

El módulo que de alguna forma se encarga de colocar en la memoria del FPGA la imagen será tratado más adelante. Se parte del punto en el cual la imagen ya está almacenada en la memoria que utilizará el FPGA. El objetivo principal de este capítulo es mostrar el diseño del módulo que genera los niveles de la pirámide gausiana para una imagen dada.

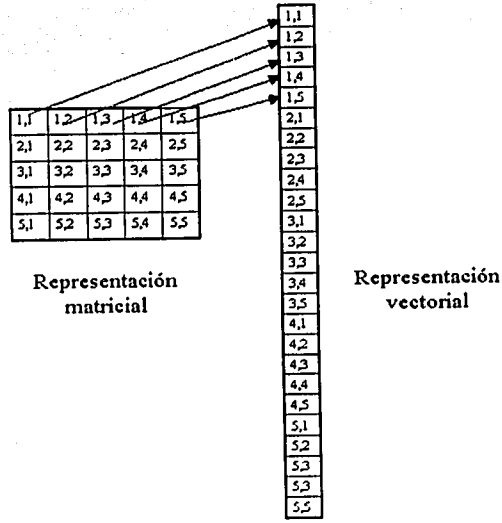

Figura 4-4: Ejemplo de la representación vectorial de una imagen de 5x5.

4.2 Objetivo de la arquitectura.

Realizar un sistema que obtenga la pirámide gausiana de una imagen en tiempo real. Este sistema debe utilizar la menor cantidad de recursos del FPGA, además de poder utilizar los resultados obtenidos para otras aplicaciones. También se deben visualizar los resultados en un monitor, para validar que nuestro diseño funcionando correctamente.

4.3 Especificaciones de la arquitectura.

Dado que esta tesis es el punto de inicio de futuros trabajos, el sistema debe basarse en los componentes disponibles en la tarjeta RC100 de Celoxica, para posteriormente tratar de diseñar una arquitectura genérica, esto es, que pueda implementarse independientemente de los dispositivos y del FPGA utilizado.

La tarjeta RC100 contiene los elementos descritos en el capítulo 3. La arquitectura utilizará las bibliotecas disponibles para el acceso al decodificador de vídeo, al DAC, a las Memorias, para resolver el problema de la generación de la pirámide gausiana.

4.4 Descripción general

Como se mencionó anteriormente, la tarjeta RC100 de Celoxica lee vídeo a través de un decodificador de vídeo y da como salida vídeo por un DAC que tiene como salida un conector VGA. Entonces, el objetivo es diseñar una aplicación que dado estos dispositivos, reciba como entrada una imagen y nos proporcione como salida los niveles de la pirámide de la imagen recibida. Los módulos principales de nuestra arquitectura se muestran en la figura 4.5.

A continuación se da una explicación muy general de los módulos de la arquitectura propuesta:

- En primer lugar, se tiene un módulo que tomara la imagen de una fuente (que en este caso es el decodificador de vídeo) y almacenará esta imagen en alguna parte de la memoria.
- También tenemos el módulo que lee la imagen de memoria y va calculando los niveles sucesivos de la pirámide.
- Finalmente, tenemos el módulo que despliega el resultado de la pirámide en la pantalla.

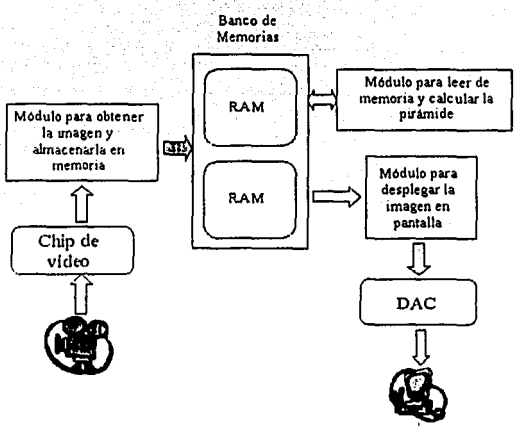

Figura 4-5: **Módulos de la Arquitectura Propuesta**

4.5 Bloques funcionales.

En esta sección se dará una explicación detallada de los módulos planteados en la sección anterior.

4.5.1 Módulo que lee del decodificador y guarda los datos a memoria

Este módulo se encarga de leer el vídeo a partir del decodificador de vídeo y almacenarlo en uno de los bancos de memoria SSRAM. El decodificador nos entrega el vídeo en tramas de 33-bits, en donde vienen almacenados los datos.

1 bit	16 bits	16 bits
Bit de control	Datos	
33 bits		

Figura 4-6: Formato de la trama entregada por el decodificador de vídeo.

El decodificador utiliza un formato RGB para la codificación de la imagen de salida. Esto quiere decir que se descompone la imagen en cada uno de sus tres componentes (R, G y B). El decodificador utiliza 16 bits para codificar un píxel. La distribución de los píxeles utilizados para cada componente se muestra en la figura 4.6.

5 Bits	6 Bits	5 Bits
R	G	B
16 Bits		

Figura 4-7: Distribución de los componentes RGB dentro los 16 bits utilizados para codificar un píxel

Entonces, de los 16 bits utilizados para codificar un píxel, 5 son utilizados para el componente R, 6 para el componente G y 5 para el componente B.

El decodificador de vídeo no siempre nos entrega píxeles, sino que depende del valor del bit de control ubicado al principio de la trama. Este bit de control nos indica si la trama es un conjunto de datos o un comando o token. Si el bit de control tiene el valor de 1, la cadena es un conjunto de datos. Si el bit de control tiene el valor de 0, entonces la cadena es un comando. Vale la pena mencionar que para efecto del direccionamiento a memoria de los píxeles recibidos, utilizamos dos variables de ámbito local en este proceso. Estas variables son el contador de línea y el contador de píxel.

Ahora, cuando la cadena sea un comando, dependerá de que tipo de comando sea. Entonces, hay que comparar esta subcadena para identificar el tipo de comando del que se trata. Si la cadena es un comando de inicio de cuadro, quiere decir que se esta comenzando a tomar un nuevo cuadro (ya sea el cuadro inicial o simplemente un nuevo cuadro). Entonces, los contadores tanto de píxel como de línea deben de ser puestos a cero. En caso de que sea un comando de inicio de línea, sólo el contador de píxel deberá ponerse a cero, mientras que el contador de línea se incrementará en una unidad

Cuando la cadena sea un conjunto de datos, deberán tomarse otras acciones. Primero, deberá almacenarse en memoria los datos obtenidos del decodificador. La dirección en donde será almacenado la pareja de píxeles dependerá de los valores del contador de línea y del contador de píxel, así como de un determinado valor de offset, el cual nos indica a partir de que posición de la RAM comienza a almacenarse la imagen. Debido a que la palabra de la memoria SSRAM es de 32 bits, podemos almacenar los dos píxeles en una misma localidad. Posteriormente, se incrementará el contador de píxel. Debido a que el decodificador de vídeo nos provee de pares de píxeles, el incremento de este contador será de dos en dos.

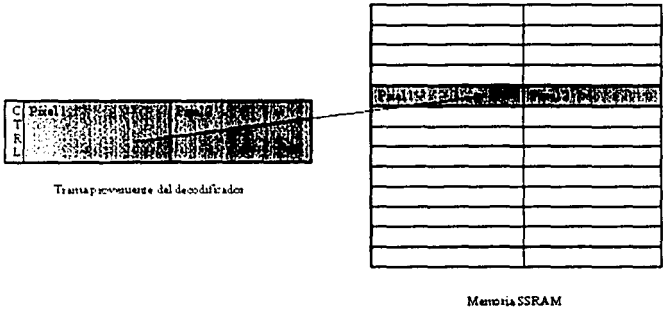

Figura 4-8: Debido a que los datos entregados por el decodificador tienen una longitud de 16 bits y el tamaño de palabra de la SSRAM es de 32 bits, es posible guardar dos píxeles en una misma localidad de memoria en el mismo ciclo de reloj.

Para ilustrar el funcionamiento del módulo, se diseño un autómata en el cual se muestran los estados y las acciones que serán llevadas a cabo en función de las entradas, que son los datos enviados en la trama por el decodificador. El estado inicial del autómata es el inicio de la operación del decodificador. En el paso del estado 1 al estado 2 es provocado porque el decodificador envía un comando de inicio de cuadro. Entonces, las acciones tomadas consisten en poner el contador de línea y el contador de píxel a cero. En el paso del estado 2 al estado 3, el decodificador envía un comando de inicio de línea, entonces sólo el contador de línea se pone a cero. En el paso del estado 3

al estado 4, el decodificador recibe datos, los cuales serán almacenados en la memoria. El autómata permanecerá en el estado 4, tomando datos de y almacenándolos en su respectiva posición de memoria, hasta que llegue o un comando de inicio de cuadro o un comando de inicio de línea, entonces, se producirá una de las siguientes acciones:

1) Si el comando recibido es un token de inicio de cuadro, el autómata regresará al estado 2, y las acciones a realizar serán reiniciar tanto el contador de línea como el contador de píxel .

2) Si el comando recibido es un token de inicio de línea, el autómata regresará al estado 3, y las acciones a realizar serán incrementar el contador de línea en una unidad, mientras que el contador de píxel será reiniciado.

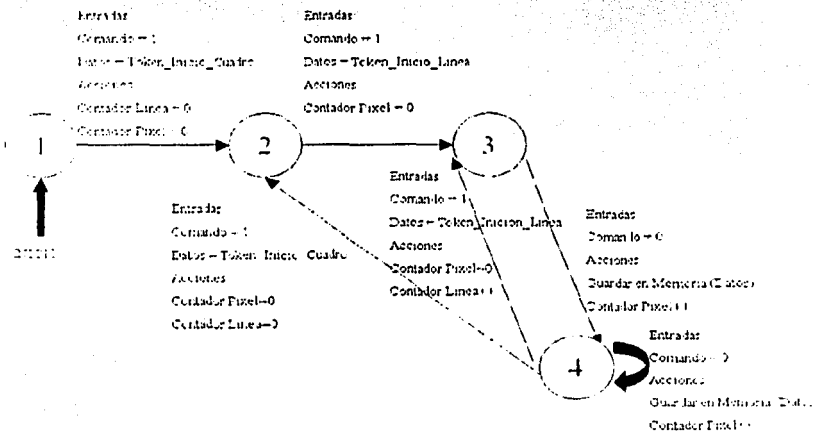

Figura 4-9: Autómata que describe el proceso de captura de la imagen.

4.5.2 División de Módulos

Diseñamos un par de módulos secuenciales que efectúen tanto el cálculo de la pirámide gausiana de una imagen como el proceso de desplegarla en pantalla en un tiempo menor o igual al tiempo de barrido sobre toda la imagen. Entonces, lo que deseamos que dada una imagen que esta almacenada en memoria, obtener todos los niveles de su pirámide y desplegarlos en N*M ciclos de reloj. Tenemos la ventaja de que las imágenes obtenidas en el proceso de obtención de la pirámide no van a ser más grandes que la imagen original, sino que cumplirán la propiedad de que cada imagen tendrá cuatro veces menos líneas y píxeles que la predecesora, permitiéndonos llevar a cabo nuestro procesamiento.

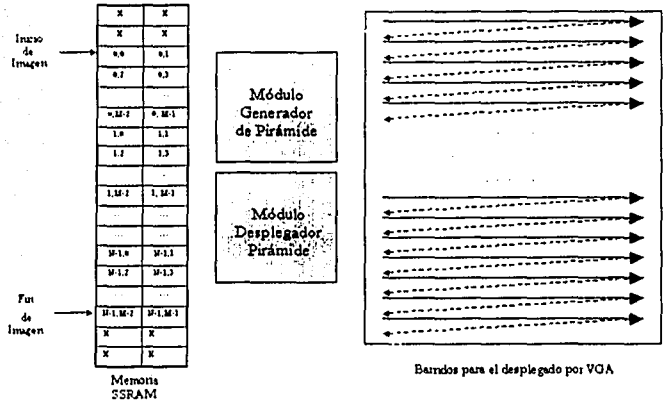

Figura 4-10: Representación de una imagen en la memoria SSRAM y despliegado de los barridos hechos para el despliegue de una imagen.

A continuación mostramos un ejemplo de las características de las imágenes resultantes. Supongamos que la imagen original tiene una resolución de 480 líneas por 512 píxeles, entonces los niveles sucesivos de la pirámide tendrán cuatro veces menos

líneas y píxeles que la original, esto es, el nivel uno tendrá una resolución de 240 líneas por 256 píxeles, el nivel dos tendrá una resolución de 120 líneas por 128 píxeles, y así sucesivamente hasta llegar al la menor resolución que se puede tener (un línea por un pixel). Para efectos del procesamiento, considero que llegar hasta el quinto nivel de la pirámide es un buen resultado.

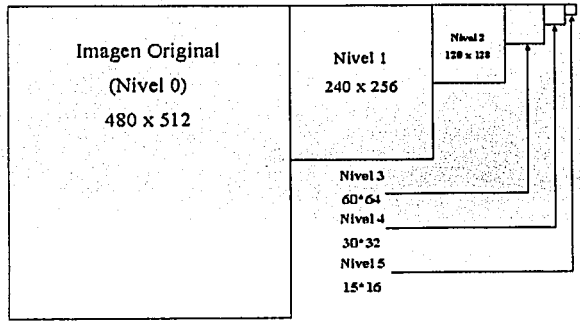

Figura 4-11: Resoluciones de todos los niveles de la pirámide gausiana considerando que la imagen original tiene una resolución de 480 líneas por 512 píxeles.

Para poder realizar esta tarea, necesitamos dividir todo este proceso en dos módulos por separado. Un módulo se encargará de calcular los niveles de la pirámide a partir de la imagen original, mientras que otro se encargara de desplegar cada uno de los nivel de la pirámide por la salida VGA. No podemos paralelizar estos módulos, dado que ambos hacen referencia al mismo banco de memoria SSRAM y ambos trabajan con las variables relacionadas al controlador de vídeo. También tendremos que ver a nuestros módulos como secuenciales uno con respecto al otro. Esto quiere decir que el módulo que despliega la imagen no puede comenzar hasta que el módulo que calcule la pirámide haya terminado sus operaciones.

Como se mencionó en la sección 3.7 del capítulo 3, el controlador de vídeo obtiene información de sus parámetros de control mediante una estructura en donde están encapsuladas un conjunto de variables. Las variables de salida son ScanX y ScanY, las cuales se van actualizando en cada ciclo y nos dan información relacionada con la coordenada del píxel que está siendo desplegado en el ciclo actual. La variable de entrada Output es en donde podemos poner el valor del píxel que queremos que se despliegue en el ciclo actual. De esta forma, tenemos el control de lo que queremos desplegar, pero no podemos controlar cuando lo queremos desplegar.

Para facilitar la comprensión de los procesos, dividimos el tiempo dedicado a nuestros módulos en dos partes iguales. Entonces, ocuparemos los primeros (N*M/2) ciclos para calcular nuestra pirámide y los segundos (N*M/2) ciclos para desplegar las imágenes resultantes. Esto quiere decir que mientras SCANY sea menor que la mitad de nuestra imagen, se ejecutará el proceso de calculo de la pirámide. Una vez que SCANY sea igual a la mitad de nuestra imagen, comenzará el proceso de desplegado de la pirámide.

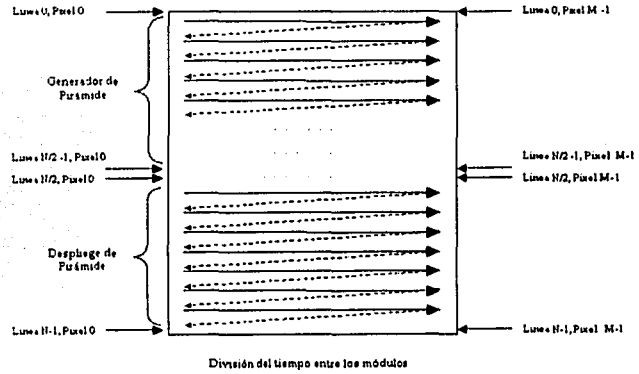

Figura 4-12: División de los tiempos de operación de cada uno de los módulos

Se puede llevar a cabo el cálculo de los niveles de la pirámide gausiana en menos de la mitad de los ciclos de barrido de la imagen, debido a la organización de la memoria SSRAM. Dado que la memoria SSRAM tiene localidades de 32 Bits, cada palabra de la memoria SSRAM puede almacenar dos píxeles. Estos dos píxeles serán utilizados durante la etapa de procesamiento. Ahora, esta propiedad puede considerarse como una ventaja y una desventaja a la vez. Es una ventaja porque nos permite leer dos pixeles en un sólo ciclo de reloj y esto es un factor para reducir el tiempo de procesamiento y es una desventaja porque en caso de que sólo se desee desplegar la imagen, necesitaríamos acceder dos veces a la misma localidad de memoria: una vez para leer el pixel par y otra vez para leer el píxel impar. La discriminación del pixel que será desplegado es llevada a cabo a partir del valor del bit menos significativo del registro ScanX, que denotaremos como ScanX[0]. Si ScanX[0] = 0, entonces el pixel es par y por tanto será desplegado el pixel almacenado en los primeros 16 bits de nuestro registro, en caso contrario, el píxel es impar y será desplegado el píxel almacenado en los segundos 16 bits de nuestro registro.

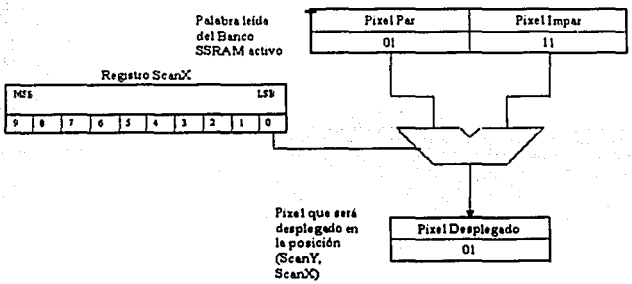

Figura 4-13: Multiplexado de las dos píxeles almacenados en un registro temporal a partir del último bit del registro ScanX que indica cual de los píxelos será desplegado.

4.5.3 Descripción Genérica de la obtención de un nivel de la pirámide

Para calcular el nivel de la pirámide bajo el esquema de lectura de SSRAM implementado en la tarjeta RC100, necesitamos leer localidades de memoria, obtener sólo el píxel par de cada palabra (el primer píxel) y almacenar cada uno de estos píxeles en un registro temporal. El primer píxel de la primera localidad será almacenado en la primera parte de un registro temporal, mientras que el primer píxel de la segunda localidad será almacenado en la segunda parte del mismo registro temporal. Una vez que se ha completado los 32 bits que forman el registro, se almacena en otra localidad de memoria. Estos dos píxeles serán los dos primeros píxeles del siguiente nivel de la pirámide. Este mismo módulo puede ser utilizado para calcular el siguiente nivel a partir del nivel actual, asumiendo ahora que el siguiente nivel de la pirámide pasará a ser el nivel actual y a partir de este nivel actual se obtendrá el siguiente nivel.

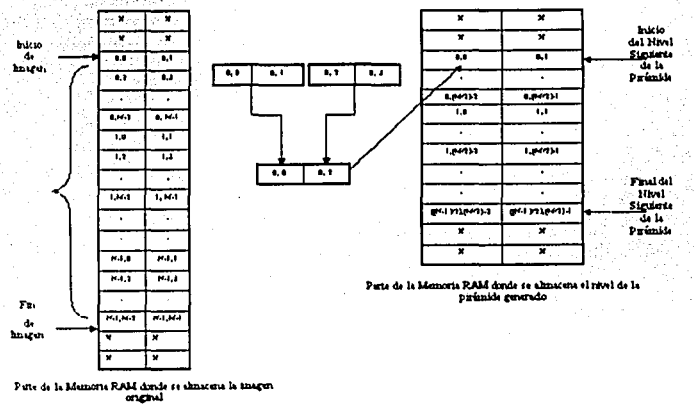

Figura 4-14: Esquema para obtener el siguiente nivel de la pirámide gaussiana a partir de una imagen dada.

4.5.4 Ejemplo de generación de pirámide para una imagen de 8x8

Para este pequeño ejemplo, se supone que la imagen original está almacenada en alguna parte de la memoria actual. El tamaño de esta imagen es de 8 líneas por 8 píxeles.

Entonces, como se describió anteriormente, para obtener la pirámide gausiana sin la aplicación del filtro gausiano correspondiente, se necesita hacer un muestreo de nuestra imagen de acuerdo con los siguientes criterios:

- Obtener las líneas pares de nuestra imagen original
- Obtener los píxeles pares de cada una de las líneas pares de la imagen original.

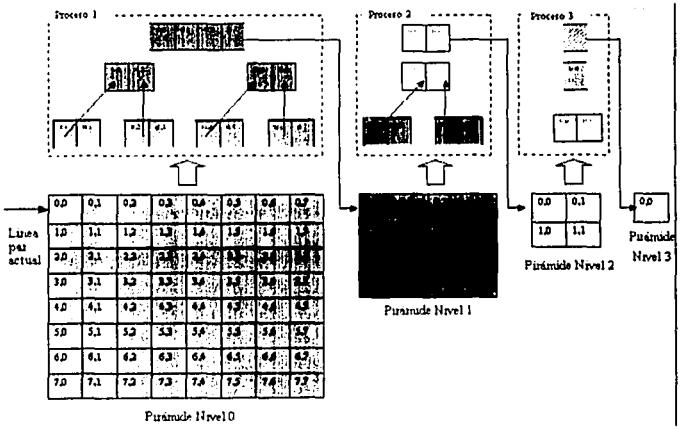

Figura 4-15: Paso 1 del proceso de obtención de tres niveles de la pirámide gausiana a partir de una imagen de resolución 8x8

Paso 1:

Se comienza con la primera línea par. Como se van leyendo por parejas de píxeles, se obtiene el píxel 0,0 y el píxel 0,1. Ahora, como solo interesan los píxeles pares, se descarta el píxel 0,1 y se almacena en un registro el píxel 0,0. Después, se lee otro par de píxeles, obteneniendo los píxeles 0,2 y 0,3 y de igual forma se descarta el pixel impar (0,3) y se almacena el pixel par (0,2). Como en este momento ya se tienen una pareja de píxeles, entonces se almacenan en una localidad de memoria RAM diferente. Estos dos píxeles serán los dos primeros píxeles del nivel 1 de la imagen. Ahora, se vuelve al proceso 1, donde se vuelven a leer otra vez dos píxeles, el 0,4 y el 0,5 y se vuelve a almacenar el pixel 0,4. Se vuelve a leer los siguientes dos píxeles y se almacena el pixel 0,6, y se tienen otra vez un par de píxeles, los cuales se almacenan en una parte de la memoria junto con los dos píxeles obtenidos anteriormente. Entonces, de un total de 8 píxeles, se han obtenido 4 píxeles como resultado del proceso. Hasta este momento, se han realizado 4 operaciones de lectura a memoria y dos operaciones de escritura.

Se tiene la primera fila del nivel 1 y como es una línea par, le toca procesamiento. Entonces, se leen de memoria los píxeles 0,0 y 0,1 y sólo se almacenan el pixel 0,0. En la siguiente iteración, se leen de memoria los píxeles 0,2 y 0,3 y sólo se almacena el pixel par. Entonces ahora se tienen un par de píxeles, los cuales se almacenarán en otra región de memoria, donde se guardará el nivel 2 de la pirámide.

Dado que se tienen en memoria solamente los píxeles 0,0 y 0,1, entonces se obtiene el pixel par que es el 0,0 y se guarda en memoria. Como ya no se tienen más píxeles que obtener del nivel anterior, entonces el pixel 0,0 sería el único integrante del nivel 3 de la pirámide. Se ha terminado con la primera línea par de la imagen.

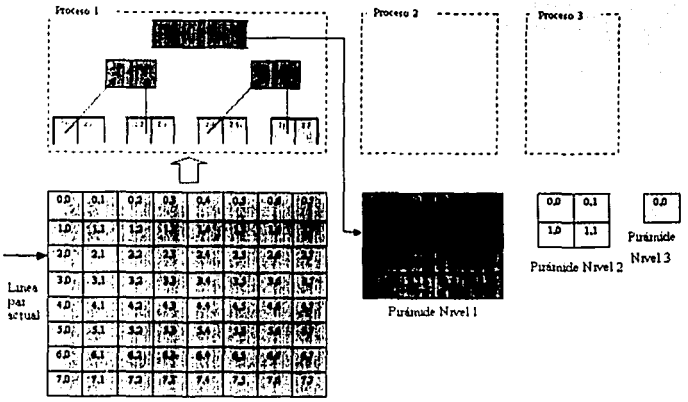

Figura 4-16: Paso 2 del proceso de obtención de tres niveles de la pirámide gaussiana a partir de una imagen de resolución 8x8

Paso 2:

Ahora se procede a leer la siguiente línea par de la imagen original. Se obtiene la línea número 2, y de la misma forma que como se hizo con la línea 0, se obtienen los 4 píxeles más representativos y se almacenan en memoria. Estos 4 píxeles más representativos pasarán a formar la segunda línea (línea 1) de la imagen del nivel 1 de la pirámide. Como esta línea no es par, no se aplicará ningún procesamiento, y por consiguiente, al no aplicarse ningún procesamiento, los otros procesos estarán inactivos. Hasta ahora, se han realizado cuatro operaciones de lectura a memoria y dos operaciones de escritura.

En general, el proceso 1 siempre trabaja obteniendo la siguiente línea par disponible, obteniendo los 4 píxeles más representativos y termina su ejecución cuando no hay más líneas pares que procesar (es decir, cuando ha terminado de leer la imagen).

El proceso 2 y el proceso 3, sólo trabajan cuando se almacena una línea par en la región de memoria que corresponde a la imagen de los niveles 2 y 3.

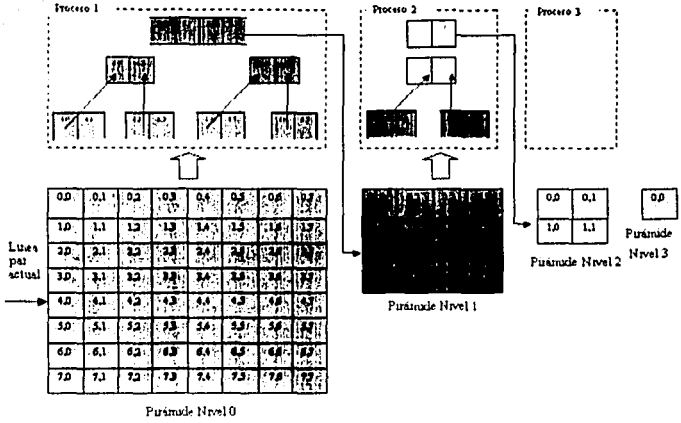

Figura 4-17: Paso 3 del proceso de obtención de tres niveles de la pirámide gausiana a partir de una imagen de resolución 8x8

Paso 3:

Ahora se va a leer la tercer línea par (línea 4). Se obtienen los 4 píxeles más representativos y se almacenan en la región de memoria correspondiente a la tercera línea del nivel uno de la pirámide. Como esta línea es par, entonces el proceso 2 obtendrá los dos píxeles más representativos y los almacenará en la región de memoria correspondiente a la segunda línea del nivel dos de la pirámide. Como esta línea no es par, entonces el proceso 3 permanecerá inactivo. Durante el procesamiento de esta línea, se han hecho 6 lecturas y 2 escrituras a memoria

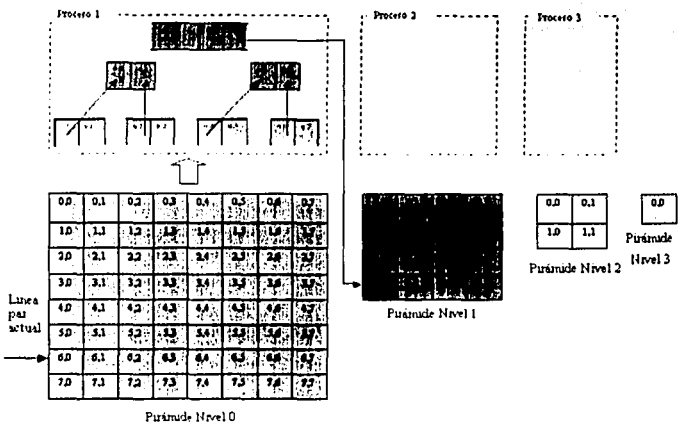

Figura 4-18: Paso 4 del proceso de obtención de tres niveles de la pirámide gaussiana a partir de una imagen de resolución 8x8

Paso 4:

Ahora, se procede a obtener la cuarta línea par de la imagen original. El proceso 1 procede a calcular la cuarta línea del nivel 1 de la pirámide, y el proceso 2 detecta que esta es una línea impar, por lo que almacena la línea en las localidades correspondientes pero no realiza ningún proceso de discriminación al no tratarse de una línea par. Para el proceso de esta línea, se efectuaron cuatro operaciones de lectura a memoria y dos operaciones de escritura.

4.5.5 Módulo Generador de la pirámide

Una vez mostrado el ejemplo para una imagen de 8 líneas por 8 pixeles, se implementará en el FPGA la misma idea, pero para una imagen de 480 líneas por 512

pixeles. Entonces, lo que se hace es dividir el tiempo en 240 líneas para el procesamiento y 240 líneas para el desplegado. Dado los requerimientos para obtener la imagen, se necesita acceder sólo a los pixeles par y a las líneas par, por lo que en vez de obtener 480 líneas, podemos leer en 240 líneas la parte de la imagen que necesitamos. Dado que se pueden leer los pixeles que se necesitan en la mitad del tiempo requerido para leer toda la línea, solo se necesitan de (512/2) ciclos de reloj para leer toda la línea.

En cada ciclo de reloj se leen de la memoria la pareja de pixeles requeridos. Como sólo se necesitan los pixeles par, siempre se almacenará la primera parte de la palabra. Una vez que se han leído dos palabras, se forma la nueva palabra, que consiste de una pareja de pixeles pares y los se almacenan en su localidad correspondiente. Esta localidad tendrá que ser calculada de la misma forma que fue calculada la localidad de donde se leyeron las palabras, pero tomando en cuenta un Offset, dado que deberá ser almacenado en una localidad superior de memoria.

Cuando ScanY = 0 y ScanX = 0, la dirección a acceder es:

$$Direccion(i,j) = i * Numero\_de\_filas + (j \setminus\setminus 1)$$
$$Direccion(0,0) = 0 * 512 + (0 \setminus\setminus 1) = 0$$

Cuando ScanY = 0 y ScanX = 1, la dirección a acceder es:

$$Direccion(i,j) = i * Numero\_de\_filas + (j \setminus\setminus 1)$$
$$Direccion(0,1) = 0 * 512 + (1 \setminus\setminus 1) = 0$$

Se puede obtener la fila del nivel uno de la pirámide en sólo 256 ciclos. Los otros 256 ciclos pueden ser utilizados para calcular los niveles restantes de la pirámide. Puesto que el número de pixeles de la imagen del nivel 1 es la mitad de los pixeles del nivel 0, podemos calcular este nivel en sólo 128 ciclos, y para el nivel 2, podemos hacer este calculo en sólo 64 ciclos, por lo para cuando el contador de ScanX llegue al M1, la primera fila de cada una de las 5 imágenes que componen a la pirámide habrán sido calculadas.

Para localizar los niveles de la pirámide obtenidos a partir de la imagen original, se procederá a realizar un direccionamiento de acuerdo con las coordenadas de la imagen. Para simplificar este proceso, lo que se hará es suponer que tenemos una imagen en memoria más grande que la original, en donde se irán almacenando los niveles de la pirámide. Dado que se calcularán solamente 5 niveles de la pirámide, entonces en la memoria habrá 5 imágenes. Podemos ver el píxel 0,0 del nivel 1 de la pirámide como el píxel 480, 0 de la imagen que tenemos en memoria. Dado que la imagen que compone el nivel 1 de la pirámide tendrá solamente 256 píxeles por línea y 240 líneas, entonces podemos ver a esta imagen como un

En si el proceso generador de la pirámide tiene que encargarse de almacenar las imágenes que componen a la pirámide en otras partes de la memoria, pero utilizando un esquema de direccionamiento que le permite al módulo que va a desplegar las imágenes acceder a estas direcciones a partir de los valores que en el momento de estarse ejecutando tengan las variables ScanX y ScanY.

El hacer uso del direccionamiento de las imágenes de la pirámide como si fueran parte de una extensión de la imagen original nos hace ver el proceso de lectura y escritura de los datos a memoria de forma más clara.

Mientras un módulo se encarga de leer la línea par de la imagen original y guardarlo en la parte de la memoria correspondiente al nivel 1 de la pirámide, otro módulo se encarga de verificar si esta línea que se está almacenando es par o impar. En caso de que esta línea sea par, entonces se encarga de llamar al módulo que genera el siguiente nivel de la pirámide. Cuando la línea que va a almacenar el módulo no es par, todos los procesos ligados a este permanecen inactivos.

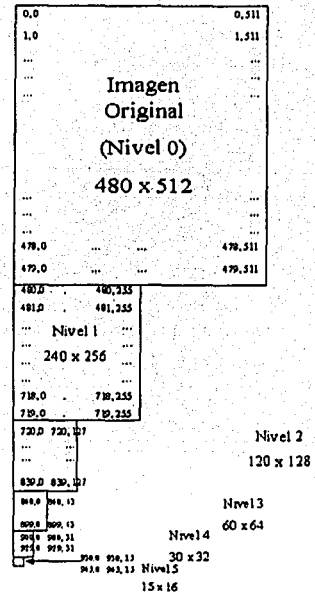

Figura 4-19: Direccionamiento de los niveles de la pirámide con respecto a las coordenadas de la imagen original

Entonces, hay un módulo seleccionador para cada nivel de la pirámide, pero en realidad estos módulos no trabajan en todas las líneas pares leídas de la imagen original, sino que depende de la paridad de la línea que estos módulos se encargan de almacenar. El módulo seleccionador 1 trabaja en todas las líneas pares de la imagen original, pero ese se encarga de activar al módulo seleccionador 2 cuando el módulo seleccionador 1 va a almacenar una línea par. El modulo seleccionador 2 almacenara esta línea, pero sólo activará al módulo 3 cuando vaya a almacenar una línea par.

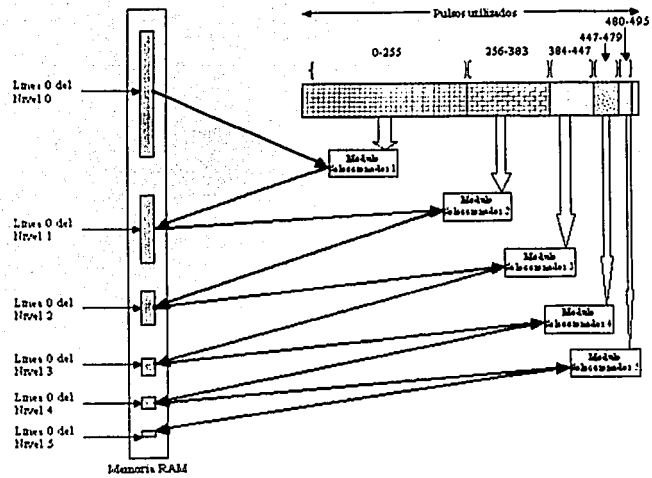

Figura 4-20: Ilustración de los módulos de selección para la fila 0 de la imagen original.

4.5.6 Módulo que despliega por VGA el resultado del procesamiento.

Una vez que ya se han generado todos los niveles de la pirámide, se procederá a desplegarla. En primera instancia, lo que necesitamos hacer es obtener las direcciones en donde están almacenadas. Como el contador ScanY tendrá el valor de 240, lo que tendremos que hacer es volver a calcular las direcciones a partir de los valores de ScanY y ScanX, pero tomando en cuenta el valor que actualmente tiene ScanY.

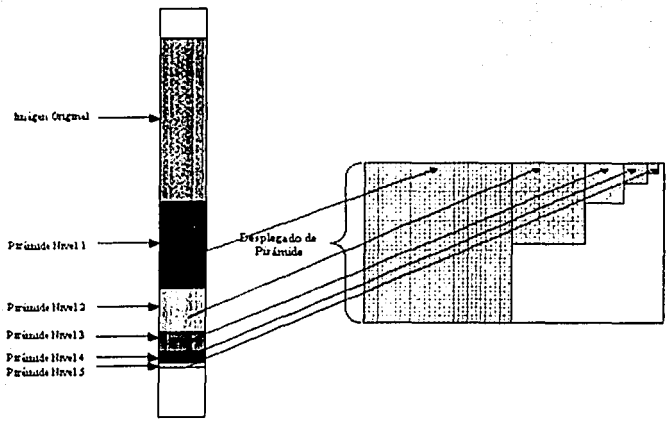

Figura 4-21: Ilustración del proceso de desplegado de las imágenes que forman la pirámide gausiana.

Dado que las imágenes que integran a la pirámide ya fueron colocadas en memoria, este módulo tiene como función principal leer estas imágenes y desplegarlas en pantalla. Ahora, para no hacer el esquema de direccionamiento muy complicado, utilizamos un direccionamiento relativo a la coordenadas de los píxeles de imagen original. Entonces, para acceder al nivel 1 de la pirámide, accedemos a las localidad 480,0, 480,1 .... 480,255, dado que esta imagen sólo tiene 256 píxeles y hasta la línea 719 ,0, 719,1 .... 719,255 dado que esta imagen sólo tiene 120 líneas, entonces 480 + 240 = 720. Para el siguiente nivel de la pirámide accedemos al píxel 720,0, 720, 1, ... 720, 128 y hasta la línea 839 (Dado que el nivel 2 de la pirámide tiene solo 120 líneas, entonces (720 + 120 = 840).

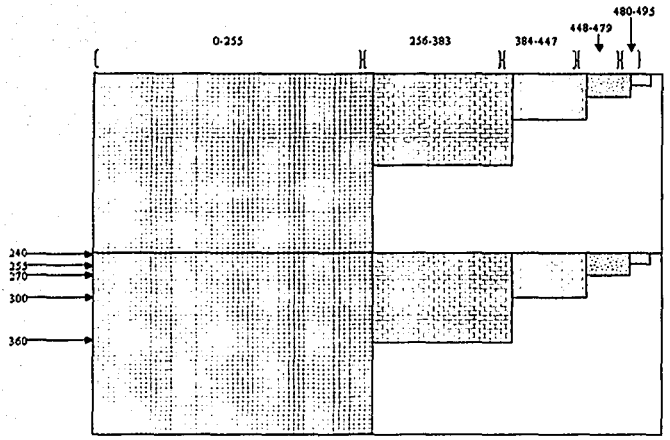

Figura 4-22: Valores de ScanX y ScanY a partir de los cuales se calcularán las direcciones de memoria en donde están localizadas las imágenes que forman parte de la pirámide.

4.5.7 Integración de todos los módulos

Una vez descrito lo que hacen cada uno de los módulos por separado, hace falta dar una explicación detallada de la integración de los módulos. Pero, como estamos diseñando un sistema que utiliza paralelismo, es difícil imaginar todos los módulos trabajando al mismo tiempo. Podríamos ver al módulo 1 trabajando en paralelo con el módulo 2 y el módulo 3. Los módulos 2 y 3 son secuenciales, esto es, primero se ejecuta uno y después se ejecuta el otro, pero el módulo 1 es paralelo a los módulos 2 y 3.

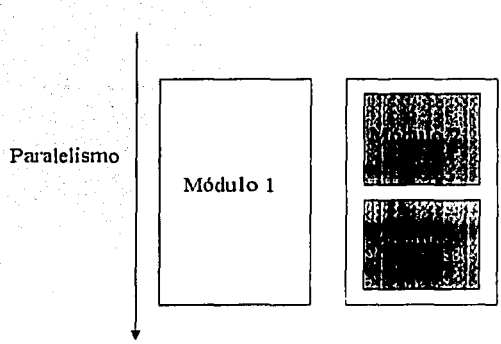

Figura 4-23: Visualización del funcionamiento en paralelos de los tres módulos que componen la arquitectura.

Los módulos 2 y 3 no pueden trabajar en paralelo porque ambos hacen uso del mismo banco SSRAM, y no se puede acceder a dos direcciones o localidades de la memoria en el mismo ciclo. Por eso es que el módulo 2 debe terminar la escritura para el que módulo 3 pueda acceder a las localidades y desplegarlas. Como el módulo 1 hace referencia a un banco distinto del utilizado por los módulos 2 y 3, entonces es posible paralelizar el módulo 1 con los módulos 2 y 3.

Mientras el módulo de lectura de imágenes está adquiriendo la imagen utilizando el decodificador de vídeo. Hay una variable global llamada ActiveRam, la cual indica a que memoria se van a guardar los datos obtenidos por el decodificador de vídeo. El cambio de esta variable ActiveRam esta controlada por el decodificador de vídeo. Cuando el decodificador de vídeo ha terminado de capturar un cuadro, cambia el valor de esta variable. Entonces, en el siguiente marco leído, se almacenará en la otra memoria SSRAM. En este momento todos los datos de la imagen original se sobrescriben, porque se supone que ya se terminó el procesamiento requerido. Una vez que se han terminado de leer los datos y viene un comando de inicio de marco, se vuelven a conmutar las memorias.

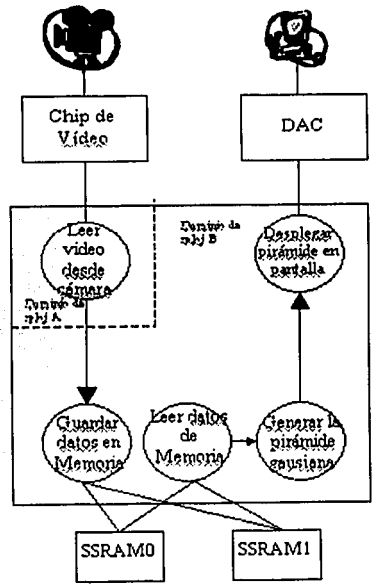

Figura 4-24: Integración de los módulos y los componentes del sistema

Capítulo 5

Implementación y Resultados

5.1. Resultados de la Simulación funcional

Para entender el funcionamiento de nuestro algoritmo, fue necesario realizar una simulación funcional de alto nivel del algoritmo. Para programar esta simulación funcional, se debe utilizar un leguaje de alto nivel, como es Lenguaje C o MATLAB. Dado las facilidades que tiene MATLAB para el manejo de archivos de imágenes, se decidió utilizar este último sobre Lenguaje C.

Básicamente, los módulos de la simulación funcional de alto nivel se muestran a continuación.

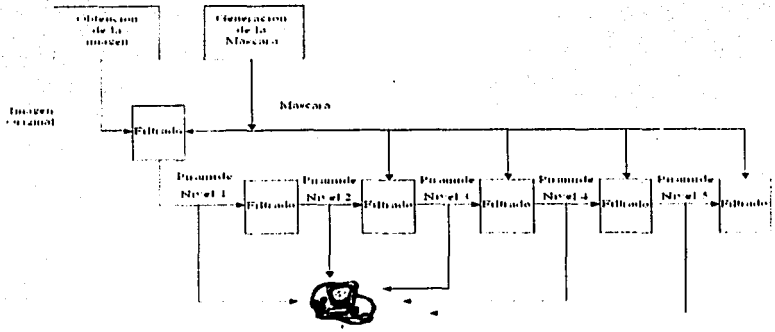

Figura 5-1: Módulos que componen a la simulación funcional de alto nivel

Los resultados obtenidos al realizar la simulación de alto nivel se presentan en la tabla 5.1.

Imagen	Nivel de la pirámide
	0
	1
	2

	3
	4
	5

Tabla 5.1: Resultados de la aplicación de la simulación funcional de alto nivel a una imagen a color

La simulación de alto nivel nos ayuda a comprender el grado de complejidad del problema a resolver. Ahora, también se debe tener cuidado en el momento de hacer esta implementación, puesto que hay una gran diferencia entre llamar a una función dada por el compilador que aplica a un filtro a una imagen y entender que es lo que hace esta función. Para efectos de esta implementación, no se utilizó ninguna función de filtrado o procesamiento proporcionada por el programa de alto nivel, sino que se intento (con éxito) entender las partes del proceso para posteriormente implementar las módulos necesitados, partiendo de la suposición que la imagen estaba en algún lugar de la memoria almacenada en forma de matriz (o matrices, tratándose de una imagen a color). Una vez hecha esta implementación y habiendo entendido todos los pormenores del problema, se procedió a realizar la implementación en el FPGA.

5.2 Descripción del proceso de prueba

Como mencionamos en el capítulo 4, a diferencia de los programas de alto nivel, los FPGAs visualizan a las imágenes como vectores en vez de matrices. Esto cambió un poco el enfoque que se le estaba dando a la implementación, pero la idea principal permaneció intacta. Lo único que tenia que modificarse en el FPGA fue el proceso de adquisición de la imagen, dado que mientras en el lenguaje de alto nivel el proceso de adquisición de la imagen consistía de una línea. Este problema en el FPGA consistió en entender como funcionaba el decodificador de vídeo (a muy alto nivel) y el tipo de direccionamiento utilizado para almacenar las imágenes.

Para probar nuestra aplicación, seguimos los pasos anteriores, y una vez que el archivo de programación había sido cargado en la memoria FLASH de la tarjeta, se conectó el Monitor y la Cámara para verificar nuestro diseño.

Ahora, para documentar el desarrollo de nuestra aplicación, tuvimos que utilizar una cámara adicional. Se utilizó una cámara para tomar la imagen original y una cámara para tomar la imagen del monitor que nos entregaba la salida resultante del proceso del pirámide.

A continuación, se muestran los resultados de bajar la programación al FPGA.

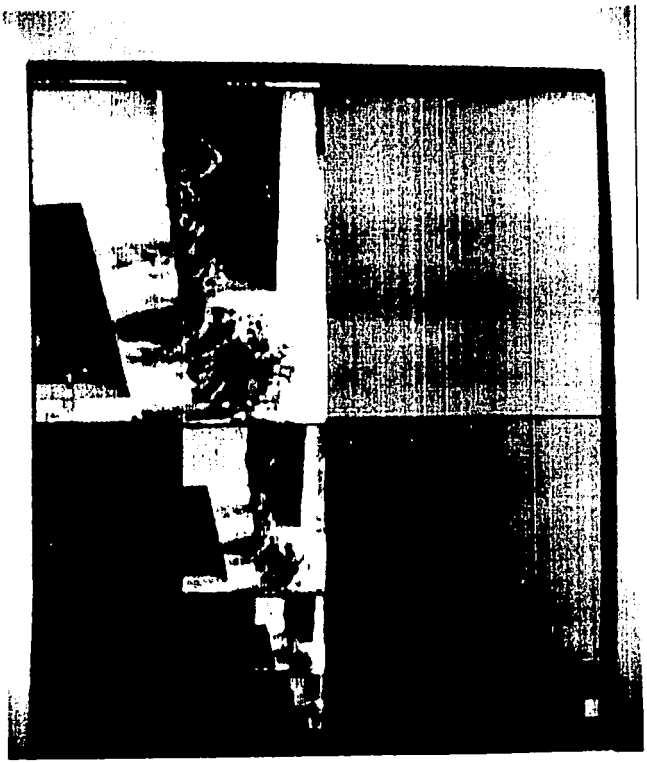

Figura 5-2: Resultado del proceso de pirámide gausiana para una imagen dada.

Figura 5-3: Resultado del proceso de pirámide gausiana para una imagen dada.

Figura 5-4: Resultado del proceso de pirámide gausiana para una imagen dada.

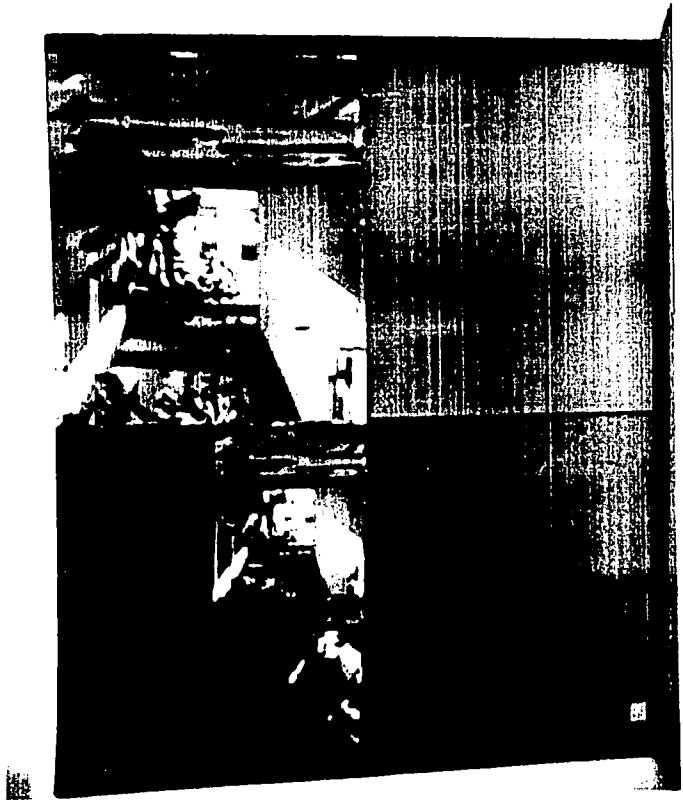

Figura 5-5: Resultado del proceso de pirámide gausiana para una imagen dada.

5.3 Comentarios acerca de los resultados

De las imágenes presentadas en la sección anterior se pueden obtener las siguientes conclusiones:

a) El resultado de la aplicación de la pirámide es óptimo ya que visualmente no se observan grandes variaciones con respecto a la imagen real. Ahora, por el momento no se ha desarrollado un método de comprobación diferente al de la visualización de la imagen, por lo que se puede concluir que los resultados son buenos.

b) Se puede observar que hay pérdida de información en los niveles más altos de la imagen. Esta pérdida de información no se debe a un error de programación, sino que se atribuye a que se tiene una menor cantidad de pixeles para almacenar la información, por lo que forzosamente debe haber pérdida de detalle. Ese es el costo que se debe pagar por trabajar con imágenes de menor resolución.

Capítulo 6

Conclusiones y Recomendaciones

Conclusiones

La arquitectura piramidal es una herramienta útil para el procesamiento de imágenes. Las ventajas de utilizar algoritmos basados en pirámides son:

- Efectúan una menor cantidad de operaciones al trabajar con imágenes de menor tamaño, y por consiguiente consume una menor cantidad de tiempo.
- Pueden trabajar con imágenes en diferentes resoluciones. Esto es muy útil cuando se utilizan módulos diferentes que obtienen información de diversas resoluciones y al final integran todos los datos en una sola imagen.

La herramienta Handel-C nos permite desarrollar una implementación hardware rápida de la arquitectura, debido a que permite desarrollar aplicaciones en términos de elementos de alto nivel, dejando los detalles de la arquitectura al compilador utilizado. Dentro de las ventajas que podemos obtener del uso de Handel-C para el desarrollo de aplicaciones son:

- El programador no tiene que preocuparse por operaciones de bajo nivel, como es la tranferencia de datos entre dispositivos, direccionamiento de datos a memoria, uso de señales de control, etc
- El tiempo de desarrollo de las aplicaciones es mucho menor, debido a que el código en Handel-C es mucho más fácil de depurar gracias a que se apega al estándar ANSI-C

- En Handel-C existen directivas que generan operaciones paralelas, que se traducen en módulos hardware paralelos, lo que provoca que en la implementación final una mejor en el rendimiento.

Las ventajas de haber utilizado una plataforma de desarrollo integrada, como la tarjeta RC100 de Celoxica, son:

- La tarjeta tiene interfaces de vídeo tanto de entrada como de salida, los resultados obtenidos pueden visualizarse inmediatamente, evitando el uso de periféricos adicionales para desplegar los resultados
- Las bibliotecas proporcionadas por la tarjeta permiten enfocarse a la aplicación que se implementará, olvidándonos de los detalles del hardware, como son los pines utilizados por cada uno de los componentes y de las interfaces entre ellos.

Desventajas

- Debido a la tarjeta en donde se implementó la arquitectura (RC100) es una tarjeta para el prototipado, se tienen algunas limitaciones en cuando a la frecuencia de operación a la que trabaja y al número de recursos disponibles.

Por lo tanto, la hipótesis planteada es válida, en función de los resultados obtenidos y del diseño y validación de la arquitectura elaborada mediante este trabajo de investigación.

Recomendaciones

Como trabajo futuro del sistema realizado se proponen los siguientes puntos:

1. Debido a que el filtrado en tiempo real consume gran cantidad de recursos hardware, es necesario investigar varias formas de

implementarlo en Hardware y evaluar cual de estas nos puede dar un mejor desempeño, con el objetivo que todo sistema de tiempo real debe cumplir: proporcionar 24 imágenes por segundo.

2. Buscar una tarjeta de desarrollo que tenga menos limitantes en cuanto a la frecuencia de funcionamiento y a los recursos disponibles. El diseño de la arquitectura puede ser muy bueno pero si los recursos del FPGA son limitados, se necesitará buscar una plataforma de desarrollo más amplia. También es necesario realizar una estandarización en cuando a la forma de medir el desempeño de la arquitectura.

3. Estudiar algunas técnicas de procesamiento de imágenes que puedan utilizar los resultados arrojados por el procesamiento piramidal. Un ejemplo de estas técnicas son el método de correlación para el seguimiento de objetos y para la formación de mosaicos de imágenes.

Referencias

[Ado1] Adobe, Technical Guides, 2002. The RGB (CMY) Color Model. <www.adobe.com/support/techguides/color/colormodels/rgbcmy.html>

[Bur1] Burt J Peter y Adelson H. Edward. *The Laplacian Pyramid as a Compact Image code*. IEEE Transsactions on Communications. Abril 1983

[Bur2] Peter J. Burt & Edward H. Aldelson. *A Multiresolution Spline With Application to Image Mosaics*. ACM Transactions on Graphics, Vol 2. No 4. Octubre 1983.

[Brun1] Brunderlin A. Motion Signal Processing. IEEE Computer Graphics and Applications, Vol 13. No 3. 1993

[Brow1] Brown, S. AND ROSE, Jonathan. *Architecture of FPGAs and CPLDs: A Tutorial*, IEEE Design and Test of Computer, Vol. 13, No. 2, pp. 42-57, 1996.

[Cel1] Celoxica Ltd, *RC100 Hardware Reference Manual*. Oxfordshire U. K, <http://www.celoxica.com>

[Cel2] Celoxica Ltd, *RC100 Function Reference Manual*. Oxfordshire U. K, <http://www.celoxica.com>

[Cel3] Celoxica Ltd, *Handel-C Language Reference Manual Version 1.0*. Oxfordshire U. K, <http://www.celoxica.com>

[Del1] De la Escalera, Arturo. *Visión por Computador: Fundamentos y Métodos*. Pentice Hall, 1998.

[Eda1] Electronic Desing Interchange Format. 2002. *About Edif*. <http://www.edif.org/about.html>.

[Green2] Greenspan H et al. *Learning Texture Discrimination Rules in a Multiresolution System*. IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol 16, No 9. Septiembre de 1991.

[Led1] Laboratorio de Electrónica Digital, Universidad Católica "Nuestra Señora de la Asunción". *Sistemas de Tiempo Real*. Asunción, Paraguay. http://www.led.uc.edu.py/str/primera.htm#introduccion

[Mer1] Meraz Ochoa, Victor Manuel. *Sistema basado en FPGA para el Desarrollo de Aplicaciones de Visión por Computadora en Tiempo Real*. INAOE, 2001

[Paj1] PAJARES, G. AND DE LA CRUZ, J. *Visión por Computador*. Ed. Ra-ma., México, 1995.

[Per1] Pérez de la Blanca, Nicolás. *Fundamentos del Tratamiento de Imágenes por Computador*. <bucanero.ugr.es/wwwcurso/ccordoba/node16.html>

[Pet1] Peter, Christian. *Overview: Hardware Compilation and the Handel-C language* Oxford University Computing Laboratory http://web.comlab.ox.ac.uk/oucl/work/christian.peter/overview_handelc.html

[Sarn1] Sarnoff Corporation. *Pyramid-Based Video Processing*. 2002. http://www.sarnoff.com/government_professional/vision_technology/core_technology/pyramid_video_processing.asp

[Son1] Sonka, Milan., et al. Image Processing Analysis and Machine Vision. Brooks/Cole Publishing Company, 1999,

[Tej1] Tejeda Yépez, Juan Carlos. *Arquitectura FPGA para filtrado de Imágenes en Tiempo Real*. INAOE, 2001

[Xil1] Xilinx, 2002. *Device type comparisons.* <http://www.fpga-faq.com/FAQ_Pages/0007_Device_type_comparisons.htm>

Apéndice 1

Tablas de Conexiones entre los pines de los componentes de la tarjeta y los pines del FPGA

Descripción	Pines del FPGA
DATA[15:0]	E20,D21,C22,B19,C18,D17,A19,B18
ADDRESS[23:0]	E15,A17,D15,C16,D14,E14,A16,C15,B15,E13,A15,F12 C14,B14,A14,D13,C13,D13,E12,A13,B12,D12,C12,D11
STS	D16
BYTE	E16
CE	B17
OE	C17
WE	A18

Tabla 3.1: Descripción de las funciones de la memoria Flash y su conexión con los pines del FPGA

Banco 0	
Función SSRAM	Pines del FPGA
P[3:0]	L5, L1, L3, L4
B3[7:0]	F1, F2, H5, G3, G4, E1, E2, F3
B2[7:0]	G5, F4, E3, D2, F5, C1, E4, B1
B1[7:0]	L6, K2, K4, K3, K1, K5, J1, J3
B0[7:0]	J2, J5, H1, J4, H2, H3, G1, H4
Address[17:0]	A0, E8, D8, C7, D7, B6, A5, D6, C6, B5, E7, A4, E6, B4, A3, B3, D5, C5
GW	D10
BW[3:0]	C8, B8, D9, A8
CLK	A7

CE	E9
OE	C9
ADSP	B9
ADV	E10

Tabla 3.2: Descripción de las funciones de la memoria SSRAM del banco 0 y su conexión con los pines del FPGA

Banco 1	
Función SSRAM	Pines del FPGA
P[3:0]	V4, V3, W3, Y2
B3[7:0]	P3, P5, R1, P4, P2, N5, P1, N4
B2[7:0]	N3, N3, M5, N1, M4, M3, M6, M1
B1[7:0]	W2, Y1, U4, V2, W1, T4, T3, U2
B0[7:0]	T5, V1, R5, U1, T2, R4, T1, R2
Address[17:0]	V11, Y8, W8, W7, AA7, AB6, AA6, V8, AB5, AA5, Y7, W6, AB4, AA4, Y6, VY, AB3, W5
GW	AA9
BW[3:0]	W9, AB9, Y9, V10
CLK	V9
CE	AB8
OE	W10
ADSP	Y10
ADV	AB10

Tabla 3.3 Descripción de las funciones de la memoria SSRAM del banco 1 y su conexión con los pines del FPGA

Función del Decodificador	Pines del FPGA
DATA[15:0]	N19,P21,P20,P19,R22,P18,R19,U22,R18,T21,V22,T19, T20, U21, W22, T18
VREF4	M20
REF	P22

CREF	M19
HS	N20
VS	N18
CE	N21
NRES	M17
RTSO	M22
LLC	M18, W12 (W12 en un buffer de reloj global que es utilizado si el dominio del vídeo es temporizado desde afuera del chip)
I2C DATA	V21
I2C CLK	U16

Tabla 3.4: Descripción de las funciones del decodificador de vídeo y su conexión con los pines del FPGA

Función de Salida de Vídeo	Pines del FPGA
Red[7:0]	K22,K21,K19,L22,L21,L18,L17,L20
Gren[7:0]	H19,H21,J19,J22,J18,J20,K18,J21
Blue[7:0]	F21,F19,F22,G19,G20,G18,G21,H18
Video DAC Clock Pin	E22
Video DAC Sync Pin	F20
Video DAC Blank pin	F18
H-Sync	E21
V-Sync	D22

Tabla 3.5: Descripción de las funciones del DAC y su conexión con los pines del FPGA