IMPLEMENTACIÓN DE FILTROS DETECTORES DE BORDES EN FPGA

ALEXANDER CORONADO VARGAS
LUIS ANDRÉS MALDONADO PARRA

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
CARRERA DE INGENIERÍA ELECTRÓNICA
BOGOTÁ D.C.
2011

IMPLEMENTACIÓN DE FILTROS DETECTORES DE BORDES EN FPGA

TRABAJO DE GRADO Nº 1032

ALEXANDER CORONADO VARGAS
LUIS ANDRÉS MALDONADO PARRA

Informe Final del Trabajo de Grado Presentado
para Optar al Título de Ingeniero Electrónico

Director:
ING. JUAN CARLOS GIRALDO CARVAJAL M.Sc.

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
CARRERA DE INGENIERÍA ELECTRÓNICA
BOGOTA D.C
2011

Advertencia

"La Universidad Javeriana no se hace responsable de los conceptos emitidos por sus alumnos en sus trabajos de tesis. Solo velará porque no se publique nada contrario al dogma y la moral católica y porque la tesis no contenga ataques o polémicas puramente personales; antes bien, se vea en ella el anhelo de buscar la verdad y la justicia".

Reglamento de la Pontificia Universidad Javeriana, Artículo 23, de la Resolución 13, de Julio de 1965

PONTIFICIA UNIVERSIDAD JAVERIANA

FACULTAD DE INGENIERÍA

INGENIERÍA ELECTRÓNICA

RECTOR MAGNIFICO: R.P. JOAQUÍN SÁNCHEZ S.J.

DECANO ACADÉMICO: ING. FRANCISCO JAVIER REBOLLEDO MUÑOZ

DECANO DEL MEDIO UNIVERSITARIO: R.P. SERGIO BERNAL S.J.

DIRECTOR DEL DEPARTAMENTO DE ELECTRÓNICA: ING. JORGE LUIS SÁNCHEZ M.Sc, M.Edu.

DIRECTOR DE CARRERA: ING. JUAN MANUEL CRUZ M.Edu.

DIRECTOR DEL PROYECTO: ING. JUAN CARLOS GIRALDO CARVAJAL M.Sc.

AGRADECIMIENTOS

Este proyecto no hubiera podido llevarse a cabo sin el apoyo incondicional que mi familia y mis amigos me brindaron. Agradezco a mi mama Alba, mi hermano Alberto y mi hermana Dianne por tanta paciencia. A mis amigos y demás personas que creyeron en este proyecto y que de alguna manera aportaron un poco durante este proceso. Para terminar un agradecimiento especial para nuestro director Juan Carlos Giraldo, al igual a nuestro asesor José Luis Uribe por su dedicación una vez más, gracias a todos.

ALEX

TABLA DE CONTENIDO

INTRODUCCIÓN 1

1. MARCO TEÓRICO 3

1.1 PRELIMINARES PROCESAMIENTO DE IMÁGENES 3

1.2 REPRESENTACIÓN DE IMÁGENES DIGITALES 3

1.3 FILTROS DETECTORES DE BORDE *SOBEL* Y *PREWITT* 5

1.4 IMPLEMENTACIÓN EN FPGA 7

2. ESPECIFICACIONES 9

2.1 ESPECIFICACIONES TARJETA DE DESARROLLO EP2C20F484C7® 9

2.1.1 FPGA *Cyclone* II 2C20 10

2.1.2 Memorias externas 11

3. DESARROLLO 12

3.1 ESPECIFICACIONES PRELIMINARES DEL DISEÑO 12

3.1.1 Descripción general del funcionamiento del sistema 12

3.1.2 Descripción entradas y salidas 13

3.2 DIAGRAMA DE BLOQUES 15

3.2.1 Adquisición de datos 16

3.2.2 Procesamiento de datos 21

3.2.3 Unidad de control 26

3.2.4 Etapa de salida 32

3.3 DIAGRAMA DE TIEMPOS 32

3.4 DESCRIPCIÓN AHPL 32

3.5 DESCRIPCIÓN VHDL 32

4. ANÁLISIS DE RESULTADOS 33

4.1 REPORTE GENERADO POR QUARTUS II DE ALTERA® 33

4.1.1 Recursos empleados por el dispositivo programable 33

4.1.2 Análisis de tiempos de propagación 34

4.1.3 Asignación de pines tarjeta de desarrollo 36

4.1.4 Diagrama De Tiempos 37

4.2 PROTOCOLO DE PRUEBAS 38

4.2.1 Imágenes de prueba 38

4.2.2 Velocidad de procesamiento 40

4.2.3 Implementación en distintas tarjetas de desarrollo 42

4.2.4 Análisis adicionales 43

5. CONCLUSIONES 45
6. FUENTES DE INFORMACIÓN 47
7. ANEXOS 48

ÍNDICE DE FIGURAS

Figura 1.1 Plano espacial de una imagen MxN. 4
Figura 1.2 Máscara 3x3. 4
Figura 1.3 Aplicación de una máscara 3x3 a una imagen. 4
Figura 1.4 Máscara de un filtro general. 5
Figura 1.5 Máscaras *Sobel*. 7
Figura 1.6 Máscaras *Prewitt*. 7
Figura 2.1 Tarjeta de desarrollo EP2C20F484C7 9
Figura 2.2 Especificaciones de *FPGA Cyclone* II®. 10
Figura 2.3a Especificaciones de tiempos de lectura SRAM. 11
Figura 2.3b Especificaciones diagramas de tiempos para lectura de memoria. 11
Figura 3.1 Diagrama general 12
Figura 3.2 Interruptores tarjeta de desarrollo 14
Figura 3.3 Tabla de verdad memoria *SRAM*. 15
Figura 3.4 Diagrama de bloques 16
Figura 3.5 Adquisición de datos 17
Figura 3.6 Matriz de imagen 640x480. 18
Figura 3.7 Organización memorias embebidas. 19
Figura 3.8a Primer píxel filtrado. 19
Figura 3.8b Segundo píxel filtrado. 19
Figura 3.8c Tercer píxel filtrado. 19
Figura 3.8a Cuarto píxel filtrado. 20
Figura 3.9 Procesamiento de datos. 22
Figura 3.10 Rotación de coeficientes de la máscara. 22
Figura 3.11 Máscaras de bloque *Mask_Option*. 23
Figura 3.12 Bloque *Operator*. 23
Figura 3.13 *Árbol de Wallace* de nueve entradas de 13 Bits. 25
Figura 3.14 Conversion de *Full Adder* a *Carry Save Adder*. 25
Figura 3.15 Máquina de estados. 28
Figura 3.16 Multiplexor de bloque *Embedded_Address*. 31
Figura 4.1 Reporte recursos utilizados generado en QUARTUS II®. 33
Figura 4.2 Reporte de retardos del bloque *Operator* en QUARTUS II. 394
Figura 4.3 Recursos utilizados bloque *Operator* generado por QUARTUS II®. 341

Figura 4.4 *Floorplan* generado por QUARTUS II® ... 35
Figura 4.5 Asignación de pines tarjeta de desarrollo ... 36
Figura 4.6 Diagrama de tiempos fin de procesamiento 27Mhz ... 37
Figura 4.7 Diagrama de tiempos fin de procesamiento 50Mhz ... 37
Figura 4.8 Diagrama de tiempos analizador lógico TLA5202B® ... 38
Figura 4.9 Imagen de prueba en escala de grises ... 39
Figura 4.10a Imagen resultante filtro *Prewitt* ... 39
Figura 4.10b Imagen resultante filtro *Sobel* ... 39
Figura 4.11 Histograma imagen original ... 40
Figura 4.12a Histograma filtro *Prewitt* ... 40
Figura 4.12b Histograma filtro *Sobel* ... 40
Figura 4.13 Tiempo transcurrido para filtrado de imágenes (ms) ... 41
Figura 4.14a Resultados FPGA *Cyclone* II® ... 42
Figura 4.14b Resultados FPG A *Cyclone* III® ... 42
Figura 4.14c Resultados FPGA *Stratix* II® ... 42
Figura 4.15 Recurso usados por las FPGA ... 43

LISTA DE ANEXOS

Anexo A. Evaluación tiempo de procesamiento en MATLAB® ... 48
Anexo B. Evaluación tiempo de procesamiento en C ... 49
Anexo C. Manual software Control panel para Altera® Cyclone II EP2C20F484C7N ... 51
Anexo D. Interfaz para pruebas ... 55

INTRODUCCIÓN

El ser humano emplea su sistema óptico para recoger información de su entorno que luego es enviada al cerebro el cual está particularmente adaptado para procesarla, analizarla y así llegar a conocer un entorno de antemano desconocido. El ojo humano es una herramienta maravillosa, sin embargo presenta ciertas limitaciones físicas que alteran la recepción de la información entre ellas están: su limitado ancho de banda, enfermedades genéticas y fenómenos como el *Match Banding*[1].

Uno de los problemas fundamentales en el procesamiento de imágenes es la detección de bordes, esta suministra información útil acerca de los límites del objeto de interés. De igual forma se emplea en la segmentación[2] de imágenes, realizando una reducción drástica de la cantidad de datos a ser procesados y al mismo tiempo preserva la información estructural alrededor de los límites del objeto a analizar. A través de los años se han venido desarrollando distintos algoritmos como son: Marr-Hildreth, Canny, Prewitt, Sobel, Roberts, Jain, entre otros.

Se escogieron los algoritmos de Sobel y Prewitt debido a que son los más implementados y analizados en los textos de procesamiento de imágenes [8][10], entre sus beneficios se encuentra la resistencia a ruido y su bajo nivel de complejidad frente a los demás algoritmos, mejorando así el desempeño en términos de velocidad sin perder precisión en la imagen obtenida [7].

El presente trabajo tiene como objetivo principal el desarrollo e implementación en *hardware* de estos dos algoritmos básicos en el procesamiento de imágenes, partiendo de los resultado obtenidos se pretende hacer una evaluación de desempeño del sistema implementado en *hardware* basándose en la utilización de recursos, velocidad y tiempo de procesamiento con el fin de contrastar los resultados con el desempeño de estos mismos algoritmos de procesamiento de imágenes implementados en *software*.

Tradicionalmente los algoritmos usados en el procesamiento de señales son implementados en *software*, este se encuentra limitado por el procesador en el cual se esté trabajado y por la cantidad de diferentes procesos que este esté realizando, buscando explorar formas alternativas en el procesamiento de gran cantidad de datos con un alto grado de paralelismo, se decidió implementar este proyecto en un *Field Programmable Gate Array* de ahora en adelante FPGA, este dispositivo

[1] *Match Banding*. Ilusión óptica que impide diferenciar el borde entre dos franjas, debido a la forma en que el sistema visual humano recibe la información de los fotoreceptores vecinos.
[2] Segmentación. Determina que grupo de pixeles hacen parte de un objeto de interés y cuales hacen parte del fondo con información secundaria.

semiconductor da una gran flexibilidad a la hora de hacer diseños digitales debido a su bajo costo y facilidad de reconfiguración de *hardware*. El desarrollo de estos dispositivos en los últimos años por parte de compañías como Altera®, Xilinx® y Lattice®, da la oportunidad de aprovechar al máximo los recursos embebidos de sus FPGA así como disponer de kits de desarrollo para la fácil implementación de arquitecturas digitales donde no se implementan operaciones complejas, pero si se procesa una gran cantidad de información de forma paralela.

El desarrollo del presente trabajo de grado está distribuido de la siguiente manera: los capítulos 2 y 3 hacen una explicación del procesamiento de imágenes y las especificaciones que requiere la *FPGA* para realizar dicho procesamiento. El capitulo 3 explica cómo se realiza la detección de bordes en *FPGA* y el capitulo 4 es el análisis de resultados con el cual se evaluó el sistema, para finalmente dejando el capitulo 5 para conclusiones.

1. MARCO TEÓRICO

1.1 PRELIMINARES PROCESAMIENTO DE IMÁGENES

La creciente necesidad del análisis automatizado de imágenes y la implementación en una amplia gama de aplicaciones, ha dado al procesamiento de imágenes la responsabilidad de desarrollar algoritmos para extraer características específicas en una imagen.

Entre la teoría de la segmentación se encuentra un método para detectar discontinuidades significativas entre dos regiones denominado detección de bordes. Intuitivamente, un borde es un conjunto de píxeles conectados que se encuentran en el límite entre dos regiones. Es decir, la detección de bordes consiste en destacar los píxeles donde se producen cambios bruscos en los niveles de intensidad. Diversos algoritmos encargados de la detección de bordes han sido tema de estudio y de gran aporte en la segmentación entre los cuales se destacan los operadores Sobel, Prewitt, Canny y kirsch. Para este trabajo de grado se seleccionaron los operadores Sobel y Prewitt debido que son los más implementados y analizados en los textos de procesamiento de imágenes [9].

Los métodos de procesamiento de imágenes se dividen en dos grandes grupos: Los de dominio espacial y dominio de la frecuencia. El termino dominio espacial se refiere al plano de las imágenes en sí, los métodos en esta categoría se basan en la manipulación directa de los píxeles de una imagen. Por otro lado, las técnicas de procesamiento en el dominio de la frecuencia se basan en la modificación de la imagen por medio de la transformada de Fourier. Para este trabajo de grado es de interés trabajar en el dominio espacial donde hay una gran variedad de aplicaciones en el tratamiento de imágenes.

1.2 REPRESENTACIÓN DE IMÁGENES DIGITALES

Una imagen de tamaño M x N se representa mediante una matriz en donde el origen es ubicado en la posición (0,0) como lo ilustra la Figura 1.1 El valor contenido en cada posición de la matriz corresponde a la tonalidad o valor del píxel en dicha posición.

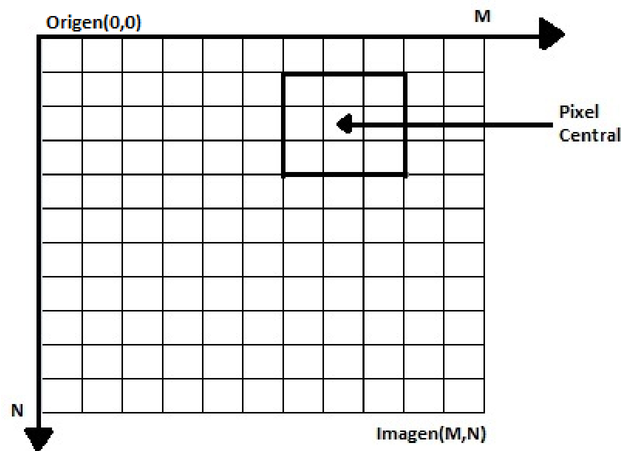


Figura 1.1 Plano espacial de una imagen MxN.

Sea un píxel p en la coordenada (x, y) de una imagen, quien tiene relacionados ocho píxeles que se encuentran ubicados alrededor de p. Este conjunto es denominado vecindario de p, donde los píxeles $(x+1, y)$,$(x-1, y)$,$(x, y+1)$ y $(x, y-1)$ son denominados vecinos directos y los correspondientes a las coordenadas $(x+1, y+1)$, $(x+1, y-1)$, $(x-1, y+1)$ y $(x-1, y-1)$ se denominan vecinos diagonales.

Para el filtrado de imágenes se deben realizar operaciones matemáticas básicas con una ventana 3x3, que comúnmente se conoce como máscara y los valores en cada posición como coeficientes.

M_11	M_12	M_13
M_21	M_22	M_23
M_31	M_32	M_33

Figura 1.2 Máscara 3x3.

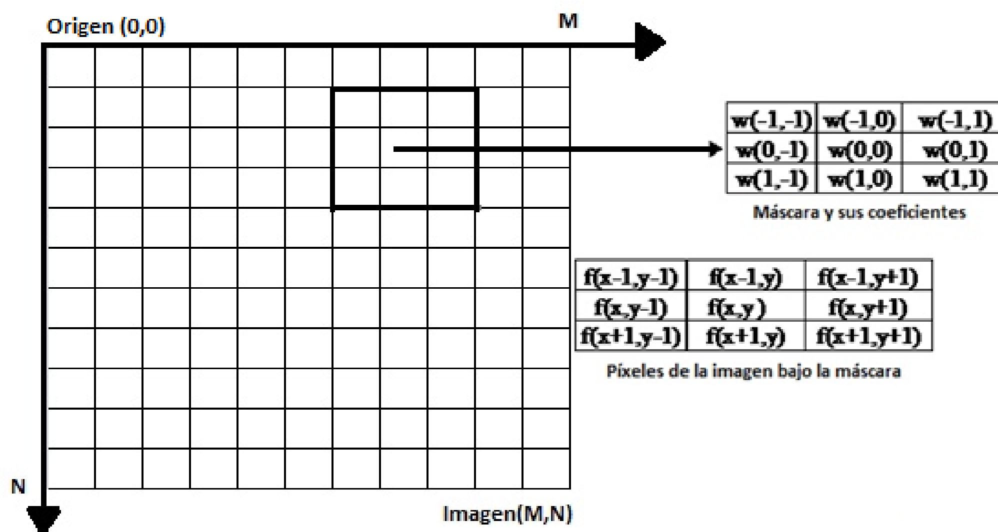


Figura 1.3 Aplicación de una máscara 3x3 a una imagen.

El mecanismo de filtrado espacial consiste en mover la máscara que se desea implementar punto por punto sobre la imagen. En cada posición (x, y), la respuesta está dada por la suma de los productos entre los coeficientes de la máscara y los píxeles involucrados o relacionados en el vecindario correspondiente. Para máscaras 3x3 el resultado R, de un filtro lineal con una máscara (x, y) es:

$$R = w(-1,-1)f(x-1,y-1) + w(-1,0)f(x-1,y) + w(-1,1)f(x-1,y+1) \\ + w(0,-1)f(x,y-1) + w(0,0)f(x,y) + w(0,1)f(x,y+1) \\ + w(1,-1)f(x+1,y-1) + w(1,0)f(x+1,y) + w(1,1)f(x+1,y+1)$$

Donde se puede observar la suma de los productos de los coeficientes de la máscara con el correspondiente píxel de la imagen. Tenga en cuenta que el coeficiente $w(0,0)$ coincide con el valor $f(x, y)$, indicando que la máscara está centrada en (x, y) y este será el valor que debe ser modificado después de pasar la máscara por esta vecindad.

En general, un filtro lineal de una imagen f de tamaño MxN con una máscara de tamaño mxn se expresa de la siguiente manera:

$g(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t)f(x+s, y+t)$

Donde, $a = (m-1)/2$ y $b = (n-1)/2$.

Para generar un filtrado completo de la imagen esta ecuación debe aplicarse para $x = 1,2, \dots, M-1$ y $y = 1,2, \dots, N-1$. De esta forma se asegura que la máscara procese todos los píxeles de la imagen.

Para una máscara 3x3 en general, la respuesta en cualquier punto (x, y) esta dada por

$$R = w_1 z_1 + w_2 z_2 + \cdots w_9 z_9 = \sum_{i=1}^{9} w_i z_i$$

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Figura 1.4 Máscara de un filtro general.

1.3 FILTROS DETECTORES DE BORDE SOBEL Y PREWITT

En la segmentación, la detección de bordes es el método más común para detectar discontinuidades en los niveles de intensidad. Estos métodos requieren la aplicación de la derivada de primer y segundo orden para cumplir con su objetivo. Como se trata de cálculos locales, para ser clasificado como borde, la transición en el nivel de intensidad asociado a un punto tiene que ser significativamente diferente al fondo en ese punto.

El cambio en el nivel de intensidad se mide por medio de la primera derivada, es decir la magnitud del gradiente de la imagen. Como una imagen $f(x, y)$ es una función de dos dimensiones, se define su gradiente como un vector de dos columnas dado por:

$$\Delta f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\delta f}{\delta x} \\ \frac{\delta f}{\delta y} \end{bmatrix}$$

Ecuación 1.1 Gradiente De Una imagen

Y la magnitud de este vector es:

$$\Delta f = mag(\Delta f)$$

$$= \sqrt{G_x^2 + G_y^2}$$

$$= \sqrt{\left(\frac{\delta f}{\delta x}\right)^2 + \left(\frac{\delta f}{\delta y}\right)^2}$$

Ecuación 1.2 Magnitud Del Vector Gradiente

La carga computacional de implementar la Ecuación 1.2 sobre toda una imagen no es trivial, y es de práctica común, aproximar la magnitud del gradiente usando el valor absoluto en lugar de la raíz cuadrada [10]:

$$\Delta f \approx |G_x| + |G_y|$$

Ecuación 1.3 Aproximación De La Magnitud Del Gradiente

Los operadores de gradiente calculan el cambio en las intensidades de niveles de gris y también la dirección en que se produce el cambio. Esto se calcula por la diferencia en los valores de los píxeles vecinos, es decir, la derivada a lo largo del eje X y el eje Y. Los Operadores gradiente requieren dos máscaras, una para obtener el gradiente de la dirección X y el otro para obtener el gradiente de la dirección Y. Estos dos gradientes se combinan para obtener una cantidad vectorial cuya magnitud representa la intensidad del gradiente en un punto en la imagen y cuyo ángulo representa el ángulo de inclinación, con el cual es posible identificar los bordes en una imagen.

Una serie de detectores de bordes, basados en la primera derivada han sido desarrollados por varios investigadores, entre los cuales se destacan los operadores Sobel y Prewitt. El operador Sobel usa una máscara de 3x3, basado en el operador gradiente.

La máscara para el operador Sobel está definida por dos ventanas como lo muestra la Figura 1.5. Las dos máscaras son aplicadas por separado en la imagen de entrada para producir dos componentes del gradiente Gx y Gy, respectivamente, en las orientaciones horizontales y verticales de la siguiente manera:

$$G_x = [f(x-1,y-1) + 2 \cdot f(x-1,y) + f(x-1,y+1)] \\ - [f(x+1,y-1) + 2 \cdot f(x+1,y) + f(x+1,y+1)]$$

$$G_y = [f(x-1,y-1) + 2 \cdot f(x,y-1) + f(x+1,y-1)] \\ - [f(x-1,y+1) + 2 \cdot f(x,y+1) + f(x+1,y+1)]$$

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Máscara Sobel Vertical Máscara Sobel Horizontal

Figura 1.5 Máscaras Sobel.

El operador Prewitt se define de la misma manera que Sobel, pero varía en los coeficientes implementados en la máscara así:

$$G_x = [f(x-1,y-1) + f(x-1,y) + f(x-1,y+1)] \\ - [f(x+1,y-1) + f(x+1,y) + f(x+1,y+1)]$$

$$G_y = [f(x-1,y-1) + f(x,y-1) + f(x+1,y-1)] \\ - [f(x-1,y+1) + f(x,y+1) + f(x+1,y+1)]$$

La figura 1.6 presenta los operadores Prewitt en su versión Vertical y Horizontal respectivamente.

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Máscara Prewitt Vertical Máscara Prewitt Horizontal

Figura 1.6 Máscaras Prewitt.

La diferencia en el operador Sobel Y Prewitt es el valor de peso dos es en uno de sus coeficientes, esto con el fin de lograr un cierto suavizado, dando más importancia a los anteriormente llamados vecinos directos. Tenga en cuenta que los coeficientes de todas las máscaras que se muestran suman 0, indicando que daría una respuesta de 0 en un área de nivel de gris constante, como era de esperar de un operador derivativo.

1.4 IMPLEMENTACIÓN EN FPGA

Tradicionalmente estos algoritmos son implementados en software el cual está limitado por el procesador sobre el cual se esté trabajado, este proyecto busca explorar formas alternativas en el procesamiento de los datos, por lo cual se decidió implementarlo en FPGA. Una FPGA es un dispositivo que contiene una matriz de celdas lógicas programables. Una celda lógica puede ser configurada para realizar una función sencilla, y los interruptores son los encargados de realizar las interconexiones entre las celdas lógicas. Un diseño personalizado puede ser aplicado mediante la especificación de la función de cada celda lógica y selectivamente establecer la conexión de cada

interruptor programable. Una vez que el diseño y la síntesis se han completado, es posible programar la FPGA con la configuración deseada y obtener el circuito a la medida. La configuración de las compuertas lógicas puede hacerse por medio del lenguaje de descripción **VHDL** (VHSIC *hardware* description language) siendo VHSIC (Very High Speed Integrated Circuit), método usado para programar dispositivos como los FPGA. El motivo de realizar este trabajo de grado sobre un FPGA, radica en la posibilidad de desarrollar una arquitectura para *hardware* de propósito específico basado en la realización de operaciones en paralelo.

2. ESPECIFICACIONES

Para el desarrollo de este proyecto se requiere una tarjeta de desarrollo con la capacidad de almacenar el archivo de la imagen original en una memoria externa a la FPGA. Además debe tener suficiente espacio de memoria embebida para guardar los píxeles involucrados durante el procesamiento de la imagen. La memoria externa debe tener la versatilidad tanto para almacenar la imagen, como el soporte de una lectura a altas velocidades. Junto con estas generalidades es necesaria una salida o interfaz grafica para observar el resultado luego de aplicar cualquier tipo de filtrado a dicha imagen. Dadas estas características necesarias, se examinaron varias tarjetas de desarrollo, considerando el costo beneficio para el proyecto se utilizó la tarjeta de desarrollo EP2C20F484C7N de ALTERA con FPGA Cyclone II 2C20, disponible en el laboratorio de electrónica.

2.1 ESPECIFICACIONES TARJETA DE DESARROLLO EP2C20F484C7®

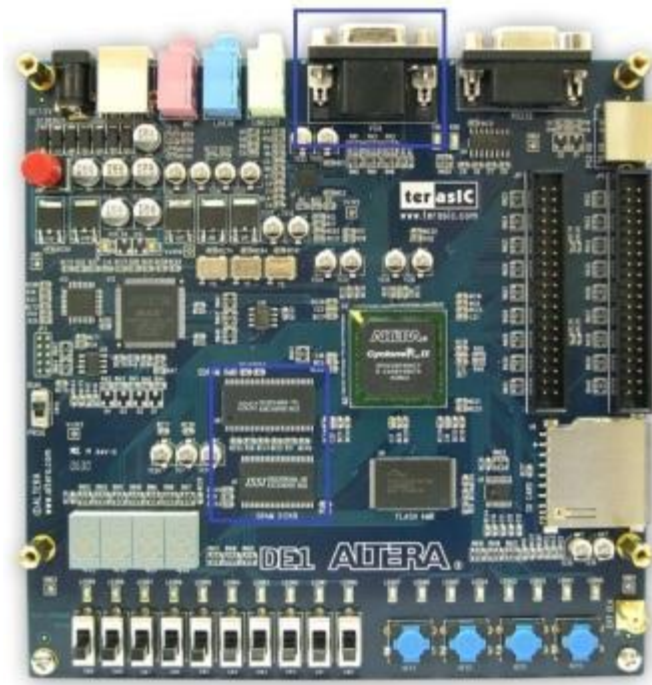


Figura 2.1 Tarjeta de desarrollo EP2C20F484C7

En la figura 2.1 tomada de la pagina *web* de ALTERA, se ve una salida VGA estándar integrada en la tarjeta de desarrollo, esta tarjeta también tiene la memoria externa a la FPGA para almacenar el archivo de la imagen. Las especificaciones generales de la tarjeta son:

FPGA:
Cyclone II EP2C20F484C7 FPGA y dispositivo para configuración EPCS4 serial.

Dispositivos de Entrada y Salida:

USB Blaster, Puerto de comunicaciones RS-232, VGA DAC (4096 colores), Puerto PS/2 para mouse teclado, Line-in, Line-out, micrófono (24-bit audio CODEC), headers de expansión (76 pines)

Memorias:

8-MB SDRAM, 512-KB SRAM, 4-MB Flash, SD *memory card.*

Switch, Keys LED, Display y Relojes:

10 toggle switches, 4 botones antirebote, 10 LEDs verdes, 8 LEDs rojos, Cuatro displays 7-segmentos, Osciladores de 27-MHz y 50-MHz, reloj externo de entrada SMA

2.1.1 **FPGA Cyclone II 2C20**

La familia Cyclone II 2C20 de Altera® es una familia de FPGA que posee 18,752 elementos lógicos (LEs), 52 bloques M4K de 4kbits de memoria, (239,616 bits que pueden ser usados como memoria RAM), 26 multiplicadores, 315 pines de entrada/salida entre otros, las características de la familia de dispositivos Cyclone II se ven en la figura 2.2 tomada de la hoja de especificaciones [1].

Feature	EP2C5	EP2C8	EP2C20	EP2C35	EP2C50	EP2C70
LEs	4,608	8,256	18,752	33,216	50,528	68,416
M4K RAM blocks (4 Kbits plus 512 parity bits	26	36	52	105	129	250
Total RAM bits	119,808	165,888	239,616	483,840	594,432	1,152,000
Embedded multipliers *(1)*	13	18	26	35	86	150
PLLs	2	2	4	4	4	4
Maximum user I/O pins	158	182	315	475	450	622

Figura 2.2 Especificaciones de FPGA Cyclone II®.

Una ventaja de utilizar una FPGA de tan altas características es la posibilidad de utilizar los multiplicadores embebidos que contiene la FPGA para realizar las operaciones en paralelo, las cuales se verán con más detalle en el siguiente capítulo. Con respecto a la memoria embebida la arquitectura requiere cargar tres filas de la imagen para ser procesadas sin perder información importante durante el desarrollo. En total se requieren almacenar 640 columnas por 3 filas con 8bits de intensidad para cada píxel, es decir:

$$640\ pixeles \times 3 filas \times 8\ bits = 15{,}360\ bits\ de\ memoria\ RAM\ embebida$$

Ya que la tarjeta de desarrollo cumple con los requisitos de memoria y multiplicadores embebidos necesarios para el proyecto, la siguiente consideración que se debe tener en cuenta es la memoria externa necesaria que se estudiará a continuación.

2.1.2 Memorias externas

Para la selección de la memoria externa, se restringió el proceso de filtrado a imágenes en escala de grises de 256 niveles, entonces el bus de la memoria debe ser de al menos 8 bits. Igualmente debe tener el espacio suficiente para almacenar la matriz imagen de 640x480, dando un parámetro mínimo en memoria de $640 \times 480 \times 8 = 2457600\, bits.$ Además de esto se debe tener en cuenta la frecuencia del reloj con la que la tarjeta de desarrollo cuenta, para este caso es de 50Mhz.

Con estos criterios y la facilidad de manejo frente a las demás memorias de la tarjeta (FLASH y SDRAM), se decidió utilizar la memoria SRAM IS61LV25616, contenida en la tarjeta de desarrollo con un bus de 16 bits y 4,194,304 bits de espacio, la cual cumple los requisitos para este proyecto.

Con las especificaciones de los ciclos de lectura de la memoria, se puede observar que el *Address Access time* es de máximo 15ns, por lo tanto es posible acceder a la memoria al menos una vez por ciclo de reloj asumiendo un reloj de 50Mhz y direcciones sincronizadas con este reloj, a continuación en la figura 2.3 se presentan las especificaciones y tiempos que emplea la memoria SRAM.

Symbol	Parameter	-10 Min.	-10 Max.	-12 Min.	-12 Max.	-15 Min.	-15 Max.	Unit
t_{RC}	Read Cycle Time	10	—	12	—	15	—	ns
t_{AA}	Address Access Time	—	10	—	12	—	15	ns
t_{OHA}	Output Hold Time	3	—	3	—	3	—	ns
t_{ACE}	$\overline{CE}$ Access Time	—	10	—	12	—	15	ns
t_{DOE}	$\overline{OE}$ Access Time	—	4	—	5	—	7	ns

Figura 2.3a Especificaciones de tiempos de lectura SRAM.

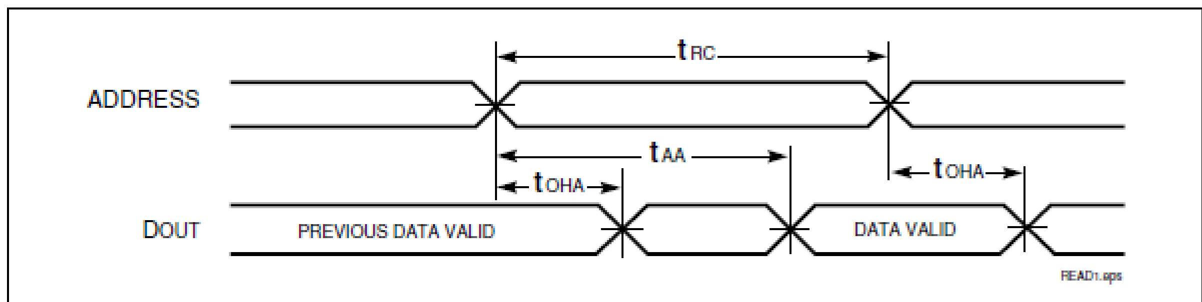


Figura 2.3b Especificaciones diagrama de tiempos para lectura de memoria SRAM.

3. DESARROLLO

3.1 ESPECIFICACIONES PRELIMINARES DEL DISEÑO

3.1.1 Descripción general del funcionamiento del sistema

El sistema propuesto a nivel general se puede observar en la figura 3.1, donde el bloque *filter* se refiere al sistema implementado en la FPGA, este sistema tiene 5 entradas, de las cuales dos son señales internas a la tarjeta de desarrollo, que son CLK y SRAM_DATA, las otras tres señales de entrada son: RESET, SOBEL y PREWITT, las cuales son externas a la tarjeta, controladas por el usuario, por medio de una tecla y dos interruptores respectivamente

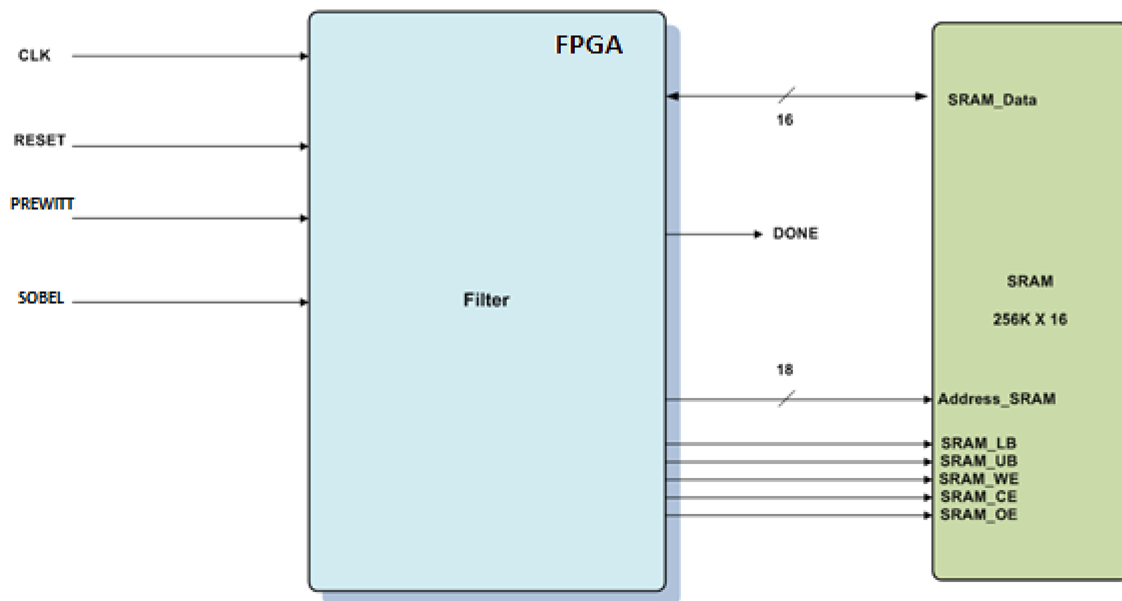


Figura 3.1 Diagrama general

El bloque *filter* se comunica por medio de los pines I-O de la FPGA con la memoria SRAM para así por medio de las entradas de control y del bus de datos de la memoria, acceder a la imagen que se quiere filtrar, la información de la intensidad de los píxeles de dicha imagen se encontrará en las primeras 153.600 posiciones de memoria.

3.1.2 Descripción entradas y salidas

Este sistema está compuesto por cuatro entradas, seis salidas y una entrada-salida (bidireccional), estas son:

Nombre	Tipo	Ancho (bits)
SRAM_DATA	IN-OUT	16
CLK	IN	1
RESET	IN	1
SOBEL	IN	1
PREWITT	IN	1
DONE	OUT	1
ADDRESS_SRAM	OUT	18
SRAM_WE	OUT	1
SRAM_CE	OUT	1
SRAM_OE	OUT	1
SRAM_LB	OUT	1
SRAM_UP	OUT	1

Tabla 3.1 Entradas y Salidas del Sistema

SRAM_DATA: Con un bus de datos de 16 bits esta señal se convierte en el eje principal de la conexión entre la FPGA y la memoria SRAM, por medio de este bus el sistema recibe la información de los 307.200 píxeles de la imagen original y es también por este bus que el sistema envía la información resultante del filtrado de la imagen hacia la memoria SRAM para remplazar los valores de los píxeles en la imagen original exceptuando los píxeles en los bordes de la imagen.

CLK: Lleva la información del reloj con el cual trabajará el sistema, esta entrada será variable y dependerá del reloj del cual provenga, para este caso la señal CLK tendrá una frecuencia de 50Mhz, este es el reloj más rápido ofrecido por la tarjeta altera usada en el proyecto. Esta señal es igual para todos los bloques por lo cual su descripción posterior será obviada.

RESET: Es la señal que da la orden de inicio a la máquina de estados del sistema, en la tarjeta de desarrollo esta tecla es la número 3 como se aprecia en la Figura 3.2. Si el usuario en cualquier momento durante el desarrollo del filtrado presiona la tecla RESET, el sistema junto con todos los contadores vuelven a su estado inicial y el proceso comienza otra vez, es de aclarar que el proceso puede llegar a ser muy rápido dependiendo del reloj usado y el usuario puede no llegar a ser capaz de dar la orden de RESET antes que ya esté finalizado el filtrado de la imagen original. Al igual que para

la señal CLK, el análisis de Reset será obviado para la descripción de los bloques individuales, la cual se realizará más adelante en este informe.

SOBEL y PREWITT: El sistema le da la opción al usuario de escoger el filtro con cual desea operar la imagen, las dos opciones que se tienen son: Sobel y Prewitt, los cuales se escogerán mediante los interruptores 9 y 8 de la tarjeta de desarrollo, como se observa en la figura 3.2, se habilita la opción deseada, subiendo el interruptor respectivo en la tarjeta de desarrollo.

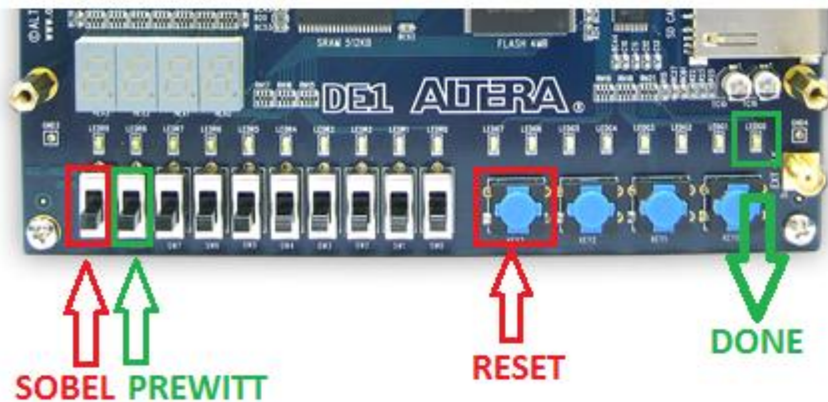


Figura 3.2 Interruptores tarjeta de desarrollo

DONE: La señal DONE encenderá el LEDG0 de la tarjeta de desarrollo como indicación de que el sistema ha terminado de filtrar la imagen. La ubicación de este *led* en la tarjeta se observa en la figura 3.2.

ADDRESS_SRAM: Se conecta entre la FPGA y la memoria SRAM, como su nombre lo indica, esta señal (interna a la tarjeta de desarrollo) con un bus de 18bits, es la encargada de transmitir la información de la dirección a la cual se quiere acceder en la SRAM.

SRAM_WE, SRAM_CE y SRAM_OE: Estas tres señales de un bit cada una, son las encargadas de controlar la memoria SRAM, como se aprecia en la figura 3.3, las señales SRAM_CE y SRAM_OE pueden estar en bajo (L) tanto para leer como para escribir, por esto estas dos señales salen del sistema con un valor bajo en todo momento, la única señal que varia es SRAM_WE siendo un alto (H) mientras se hace la lectura de la memoria SRAM y un bajo (L) para la escritura de la misma memoria.

TRUTH TABLE

Mode	$\overline{WE}$	$\overline{CE}$	$\overline{OE}$	$\overline{LB}$	$\overline{UB}$	I/O PIN I/O0-I/O7	I/O PIN I/O8-I/O15	Vcc Current
Not Selected	X	H	X	X	X	High-Z	High-Z	I_{SB1}, I_{SB2}
Output Disabled	H	L	H	X	X	High-Z	High-Z	I_{CC}
	X	L	X	H	H	High-Z	High-Z	
Read	H	L	L	L	H	D_{OUT}	High-Z	I_{CC}
	H	L	L	H	L	High-Z	D_{OUT}	
	H	L	L	L	L	D_{OUT}	D_{OUT}	
Write	L	L	X	L	H	D_{IN}	High-Z	I_{CC}
	L	L	X	H	L	High-Z	D_{IN}	
	L	L	X	L	L	D_{IN}	D_{IN}	

Figura 3.3 Tabla de verdad memoria SRAM.

SRAM_LB y SRAM_UB: SRAM_DATA es el bus de información que sale y entra a la memoria SRAM, este bus tiene un ancho de 16bits, al entrar a la memoria este se divide en dos partes iguales de 8bits cada una, dando cabida a guardar dos bytes en una posición de memoria, para el correcto uso del bus de datos se usan las señales SRAM_LB y SRAM_UB de un bit de ancho cada una, las cuales controlaran los 8 bits menos significativos y los 8 bits más significativos respectivamente, sus valores de control se encuentran en la Figura 3.3, donde un bajo (L) habilita el byte seleccionado y un alto (H) lo deshabilitará.

3.2 DIAGRAMA DE BLOQUES

El diagrama general del sistema, presentó una vista global de cómo el sistema interactúa con el usuario y con el periférico (Memoria SRAM), ahora se pretende analizar a fondo el sistema implementado en la FPGA, para esto se examinará el diagrama de bloques propuesto.

Durante la realización de este proyecto se identificaron dos etapas básicas dentro del sistema, las cuales están concatenadas en un sistema *pipeline*, estas dos etapas básicas son:

- Adquisición de datos
- Procesamiento de datos

Con la necesidad de administrar estas dos etapas del diseño, aparece una sección, denominada unidad de control, encargada de gobernar todo el sistema basándose en la información proporcionada por las entradas externas, la interconexión de las tres etapas básicas de la arquitectura, se observa en el diagrama de bloques de la figura 3.4.

Cada una de estas tres etapas fundamentales será descrita con más profundidad a continuación, al igual que los bloques Out_Register y Bidir, los cuales hacen parte del almacenamiento y direccionamiento del resultado para ser transportado hacia la memoria SRAM.

3.2.1 Adquisición de datos

En esta etapa del sistema el objetivo es identificar los 9 factores de las máscaras horizontal y vertical así como la carga en una memoria RAM embebida de la intensidad de los píxeles de la imagen a filtrar, la información de los píxeles fue previamente almacenada en la memoria SRAM de la tarjeta de desarrollo.

Figura 3.4 Diagrama de bloques

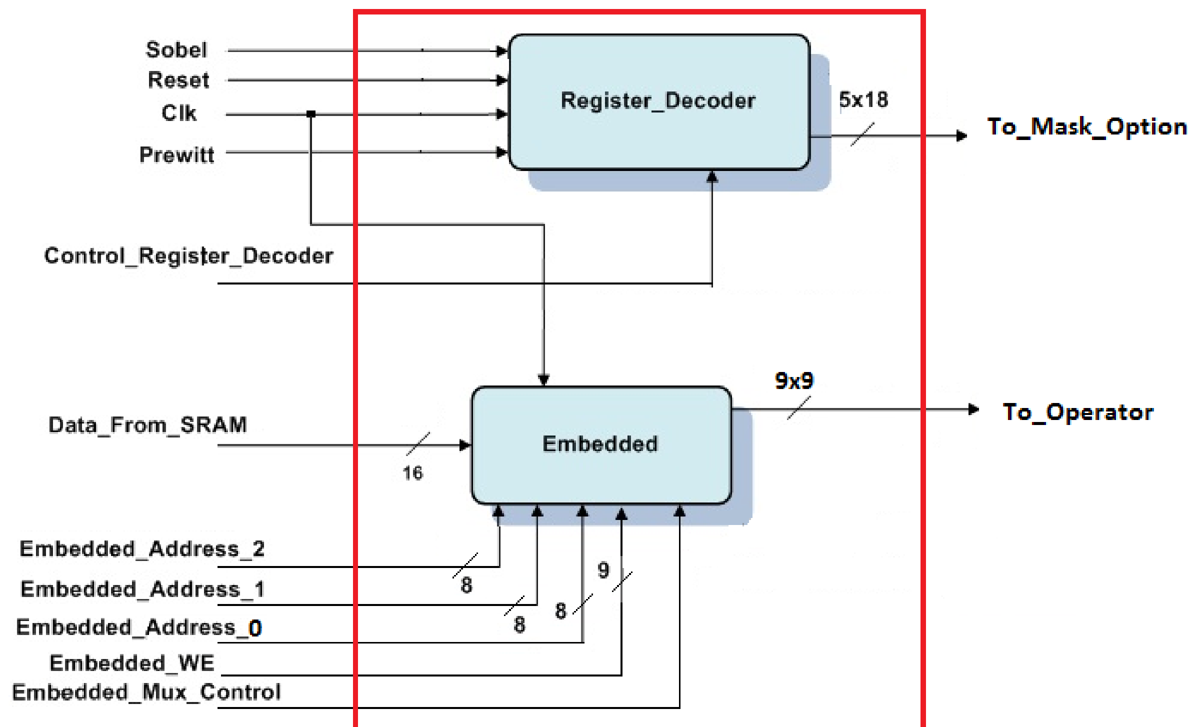


Figura 3.5 Adquisición de datos

La sección de adquisición de datos se observa en la figura 3.2, esta sección tiene 11 entradas de las cuales 3 son controladas por el usuario (Sobel, Prewitt y Reset), 2 son externas a la FPGA pero internas a la tarjeta de desarrollo (Clk y Data_From_SRAM), las 6 señales de entrada restantes provienen de la sección de control, por lo que se denominan "señales de control", estas son: Control_Register_Decoder, Embedded_Address_0, Embedded_Address_1, Embedded_Address_2, Embedded_WE y Embedded_Mux_Control; Aparte de las 11 entradas, esta sección tiene 2 salidas (To_Mask_Option y To_Operator) las cuales llevan la información de los píxeles y los coeficientes de las ventanas hacia la sección de Procesamiento de datos.

Bloque Register_Decoder

Register_Decoder es el encargado de proveer los 18 coeficientes que pasarán a conformar las máscaras horizontal (9 coeficientes) y vertical (9 coeficientes), para esto se vale de 5 entradas todas con un ancho de 1 bit y así controlar la información que será almacenada en 18 registros dentro del bloque, estas entradas son: Sobel, Prewitt, Reset, Clk y Control_Register_Decoder.

Las señales Sobel y Prewitt indican cuál de los dos filtros el usuario quiere que sea implementado, un uno lógico (Alto) selecciona el filtro y un cero lógico (Bajo) lo deshabilita, en caso dado que las dos señales tengan el mismo valor, el sistema no garantiza el resultado apropiado de ninguno de los dos filtros. Los coeficientes serán guardados en los registros cuando la señal de control

Control_Register_Decoder se encuentre en Alto, esto ocurrirá cuando la máquina de estados este en el paso seis. La salida To_Mask_Option tiene un bus de 90bits, que equivale a los 18 coeficientes, con un ancho de 5bits cada uno, esta salida transporta los coeficientes al bloque de Procesamiento de datos.

Bloque Embedded

La función de este bloque es la de cargar los píxeles que serán usados en las operaciones de filtrado, estos datos llegan al bloque por medio de la señal Data_From_SRAM de forma serial, para luego alimentar la etapa de procesamiento con la información de los 9 píxeles que se requieran en un momento dado, el enlace entre el bloque **Embedded** y la etapa de procesamiento se hace de forma paralela para así, lograr hacer el procesamiento lo más eficiente posible.

Para lograr su objetivo el bloque internamente se divide en tres módulos, cada uno de ellos compuesto a su vez por 3 memorias embebidas (RAM), para tener un total de 9 memorias RAM, las cuales tendrán la información equivalente a tres líneas de la imagen original (una línea en cada modulo), cabe recordar que la imagen para la cual se está diseñado el sistema, está compuesta por 640 columnas por 480 líneas, esta matriz de imagen se observa en la figura 3.6.

COLUMNA / FILA	1	2	3	4	5	6	...	640
1	1	2	3	4	5	6	...	640
2	641	642	643	644	645	646	...	1280
3	1281	1282	1283	1284	1285	1286	...	1920
4	1921	1922	1923	1924	1925	1926	...	2380
...	...	...	...	...	...	...	...	...
480	306561	306562	306563	306564	306565	306566	...	307200

Figura 3.6 Matriz de imagen 640x480.

La idea de guardar de a tres líneas en el bloque **Embedded** está fundamentada en que las ventanas de los filtros que el sistema implementa son de 3x3, así el sistema correrá las ventanas sobre las 3 primeras filas de la imagen, desde la primera hasta la última columna, así seguirá luego con las filas 2, 3 y 4 y continuara hasta llegar a recorrer las tres últimas filas que son las 478, 479 y 480.

Para poder almacenar los 640 píxeles de una línea en cada uno de los módulos, se tienen dos memorias RAM de 213 posiciones y una de 214 posiciones para cada modulo, cada una de las posiciones de la memoria RAM guarda un píxel (representado por 8bits), por lo que en el bloque **Embedded** finalmente se encuentran 3 memorias RAM de (214x8)bits cada una y 6 memorias RAM de (213x8)bits, el ordenamiento de las 3 memorias RAM en los 3 módulos, se aprecia en la figura 3.7, donde se tiene a la izquierda una columna de direcciones para las memorias embebidas.

	MODULO 1			MODULO 2			MODULO 3		
Address	RAM 1	RAM 2	RAM 3	RAM 4	RAM 5	RAM 6	RAM 7	RAM 8	RAM 9
0	1	2	3	641	642	643	1281	1282	1283
1	4	5	6	644	645	646	1284	1285	1286
...	...	...	...	...	...	...	...	...	...
212	637	638	639	1277	1278	1279	1917	1918	1919
213	640			1280			1920		

Figura 3.7 Organización memorias embebidas.

Con este ordenamiento el sistema logra hacer las operaciones con los 9 píxeles de la ventana en forma paralela, ya que siempre los nueve píxeles de cada una de las ventanas requeridas para el filtrado, estarán distribuidos entre las nueve memorias embebidas

Los bordes de la imagen, no se filtrarán, estos corresponden a las filas 1 y 480 y a las columnas 1 y 640, el primer píxel en ser filtrado es el 642, el segundo píxel será el 643, siendo el tercero el 644 y así sucesivamente hasta llegar a operar el píxel 1279 con lo cual se completara el filtrado de segunda fila de la imagen, luego el sistema saltará los píxeles 1280 y 1281 pertenecientes a las columnas 640 y 1 respectivamente (Bordes), para continuar con el filtrado del píxel 1282, seguido del 1283 y así sucesivamente hasta llegar al píxel 306.560 que será el último píxel que el sistema filtre, la figura 3.8 muestra las ventanas de píxeles para cada uno de los primeros cuatro píxeles a filtrar y sus respectivas posiciones en las memorias RAM embebidas.

1	2	3
641	642	643
1281	1282	1283

Add	RAM 1	RAM 2	RAM 3	RAM 4	RAM 5	RAM 6	RAM 7	RAM 8	RAM 9
0	1	2	3	641	642	643	1281	1282	1283
1	4	5	6	644	645	646	1284	1285	1286
---	---	---	---	---	---	---	---	---	---
212	637	638	639	1277	1278	1279	1917	1918	1919
213	640	-	-	1280	-	-	1920	-	-

Figura 3.8a Primer píxel filtrado.

2	3	4
642	643	644
1282	1283	1284

	2	3	4
641	642	643	644
1281	1282	1283	1284
1921	1922	1923	1924

Add	RAM 1	RAM 2	RAM 3	RAM 4	RAM 5	RAM 6	RAM 7	RAM 8	RAM 9
0	1921	2	3	641	642	643	1281	1282	1283
1	4	5	6	644	645	646	1284	1285	1286
...	...	...	...	...	...	...	...	...	...
212	637	638	639	1277	1278	1279	1917	1918	1919
213	640	-	-	1280	-	-	1920	-	-

Figura 3.8b Segundo píxel filtrado.

3	4	5
643	644	645
1283	1284	1285

Add	RAM 1	RAM 2	RAM 3	RAM 4	RAM 5	RAM 6	RAM 7	RAM 8	RAM 9
0	1921	1922	3	641	642	643	1281	1282	1283
1	4	5	6	644	645	646	1284	1285	1286
...	...	...	...	...	...	...	...	...	...
212	637	638	639	1277	1278	1279	1917	1918	1919
213	640	-	-	1280	-	-	1920	-	-

Figura 3.8c Tercer píxel filtrado.

4	5	6
644	645	646
1284	1285	1286

Add	RAM 1	RAM 2	RAM 3	RAM 4	RAM 5	RAM 6	RAM 7	RAM 8	RAM 9
0	1921	1922	1923	641	642	643	1281	1282	1283
1	4	5	6	644	645	646	1284	1285	1286
...	...	...	...	...	...	...	...	...	...
212	637	638	639	1277	1278	1279	1917	1918	1919
213	640	-	-	1280	-	-	1920	-	-

Figura 3.8d Cuarto píxel filtrado.

Al finalizar el filtrado del píxel (642) y guardar la información resultante en la memoria SRAM, es claro viendo la Figura 3.8.b que el valor de la intensidad del píxel 1 que está almacenado en la primera posición -dirección 0- de la RAM 1 ya no es relevante puesto que no se volverá a utilizar en ninguna otra operación de nuestro sistema, la posición en la cual estaba ubicado este valor, pasará a ser cargada con el píxel 1921, que es el primer píxel de la cuarta fila, este recambio de valores en la RAM 1 se advierte en la figura 3.8.b con el valor del nuevo píxel en azul, siguiendo los mismos parámetros usados con el píxel 642, el sistema filtrara el píxel 643, una vez sea guardado en la memoria SRAM el valor correspondiente al resultado, cambiará el valor ubicado en la posición 1 de la RAM 2 por la intensidad del píxel 1922 –segundo píxel de la línea 4-, tal y como se aprecia en la Figura 3.8.c, para la siguiente operación el recambio tendrá lugar en la primera posición de la RAM 3, así sucesivamente se remplazarán los valores almacenados en las RAM 1, 2 y 3 (Modulo 1) con los píxeles de la cuarta fila, al finalizar el recambio en el modulo 1, el filtrado de la segunda fila habrá terminado igualmente,

el final del filtrado de una línea equivale a operar las últimas posiciones de las memorias en los tres módulos como se observa en la Tabla 3.1.

	RAM 1	RAM 2	RAM 3	RAM 4	RAM 5	RAM 6	RAM 7	RAM 8	RAM 9
Address	0	0	0	0	0	0	0	0	0

Tabla 3.1a Primer píxel filtrado de una Línea

	RAM 1	RAM 2	RAM 3	RAM 4	RAM 5	RAM 6	RAM 7	RAM 8	RAM 9
Address	213	212	212	213	212	212	213	212	212

Tabla 3.1b Ultimo píxel Filtrado de una Línea

Siguiendo el proceso anteriormente descrito el sistema pasa ahora a filtrar la tercera fila, comenzando con el píxel 1921, la ventana azul en la Figura 3.8.b es un ejemplo de la vecindad que se operará en el filtrado de este píxel, análogamente al modulo 1, el modulo 2 cambiará sus valores, remplazándolos por los píxeles de la línea 5. Asimismo el filtrado de la línea 4 conllevará el recambio de los datos del modulo 3 por los datos de la línea 6, así el sistema cargará una a una las 480 líneas de la imagen.

Para lograr esta dinámica de almacenamiento y distribución de datos en los módulos de memorias embebidas, la sección de control, suministra al bloque **Embedded** dos tipos de direcciones, de escritura y de lectura para cada una de las memorias embebidas RAM, estas direcciones llegarán en una misma señal de 8 bits a cada memoria RAM, pero discriminadas por posición dentro del modulo, para las memorias RAM 1, 4 y 7 que se ubican en la primera posición del modulo, la dirección llegará por medio de la señal Embedded_Address_0, para las memorias RAM 2, 5 y 8 la señal de dirección es Embedded_Address_1 y para las restantes memorias RAM 3, 6 y 9 la dirección es Embedded_Address_2.

El bloque **Embedded** adquiere los datos por medio de la señal Data_From_SRAM de 16 bits, la señal de control Embedded_Mux_Control seleccionará que byte se grabará en la memoria RAM, bien sean los 8 bits más significativos -Upper byte- con un bajo ó los 8 menos significativos –Lower Byte- con un alto; la señal de entrada restante es también una señal de control, Embedded_WE llega a cada una de las memorias RAM embebidas con un bit para habilitar su escritura con un alto en un momento dado, dándole un ancho total a la señal de (1x9)bits.

3.2.2 Procesamiento de datos

El propósito de esta sección es operar los datos suministrados por la sección de adquisición de datos, estos datos incluyen: la intensidad del píxel que se está filtrando y los píxeles que se encuentren en su vecindario (3x3) y las máscaras horizontal y vertical del filtro que se está implementando.

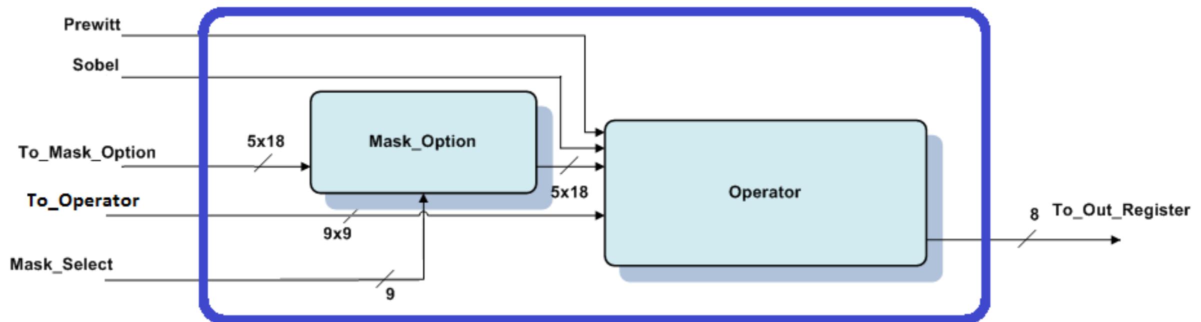


Figura 3.9 Procesamiento de datos.

Para cargar los valores de los nueve píxeles a operar en la unidad de Procesamiento de datos, se tiene la señal To_Operator con un ancho de 9x9 bits, la señal de entrada que lleva la información de los coeficientes de las máscaras es To_Mask_Option con un ancho de 18x5 bits, igualmente esta sección tiene como entradas las señales Prewitt, Sobel y la señal de control Mask_Select; la sección cuenta con una salida, To_Out_Register con ancho de 8 bits, la cual transporta la información resultante del filtrado de un píxel hacia un registro de salida, esta disposición de entradas y salidas se puede ver en la Figura 3.9.

Bloque Mask_Option

La eficiencia del sistema se basa entre otras cosas en la posibilidad de operar los nueve píxeles de un vecindario de 3x3 en paralelo con las máscaras horizontal y vertical del filtro deseado, como se vio en la sección de Adquisición de datos, los píxeles están distribuidos en nueve memorias RAM embebidas, el hecho de que la información de un píxel se encuentre almacenada en una memoria RAM sin poderse mover a otra localización, presenta un problema de rotación que se evidencia en la Figura 3.10, donde el píxel 3 para el vecindario del píxel 642 se opera con el coeficiente C de la máscara del filtro, para el vecindario del siguiente píxel (643), el píxel 3 será operado con el coeficiente B de la máscara y así mismo para el filtrado del píxel 644 el coeficiente A será el que se opere con el píxel 3.

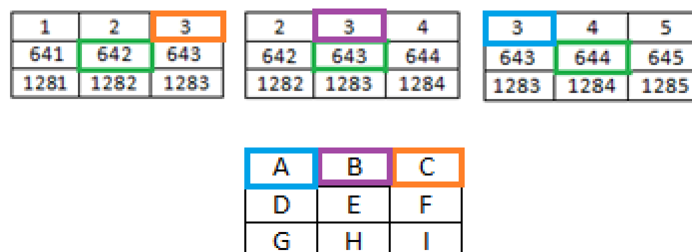


Figura 3.10 Rotación de coeficientes de la máscara.

El píxel 3 está almacenado en la memoria RAM 3 como lo muestran las figuras 3.8a, 3.8b y 3.8c por eso se necesita que los coeficientes de las máscaras roten de forma que los píxeles estén estáticos en

sus memorias RAM embebidas, esta rotación será tanto horizontal como vertical, el bloque **Mask_Option** es el encargado de realizar esta rotación, recibe 18 coeficientes de 5 bits cada uno -9 de la máscara horizontal y 9 de la vertical- por medio de la señal To_Mask_Option y entregará una de las nueve posibles máscaras para la ventana horizontal y vertical, la figura 3.11 muestra las nueve máscaras que pueden salir de la rotación.

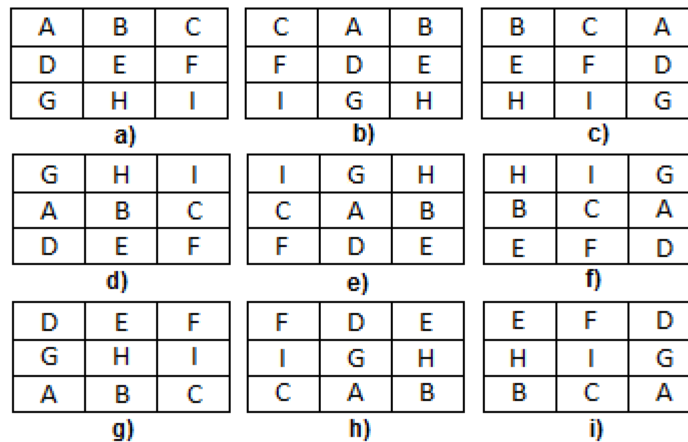


Figura 3.11 Máscaras de bloque *Mask_Option*.

Analizando las figuras 3.10 y 3.11 se deduce que para el filtrado del píxel 642 la máscara a usar será la a), para el píxel 643 será la b), para el píxel 644 será la máscara c) y para el píxel 645 volverá a ser la máscara a) completando la rotación horizontal. Análogamente para los píxeles 1282, 1283, 1284 y 1285 las máscaras serán: d), e), f) y d) respectivamente y para finalizar la rotación vertical, las ventanas restantes g), h), i) y g) serán para el filtrado de los píxeles 1922, 1923, 1924 y 1925 respectivamente. La selección de la máscara que se usara en un determinado momento está dada por la señal de control Mask_Select.

Bloque Operator

Todas las operaciones matemáticas a realizarse entre los píxeles de un vecindario y los coeficientes de las ventanas horizontal y vertical tendrán lugar dentro del bloque Operator, la subdivisión del bloque se presenta en la figura 3.12, se revelan 10 bloques internos, cada uno encargado de una operación en particular.

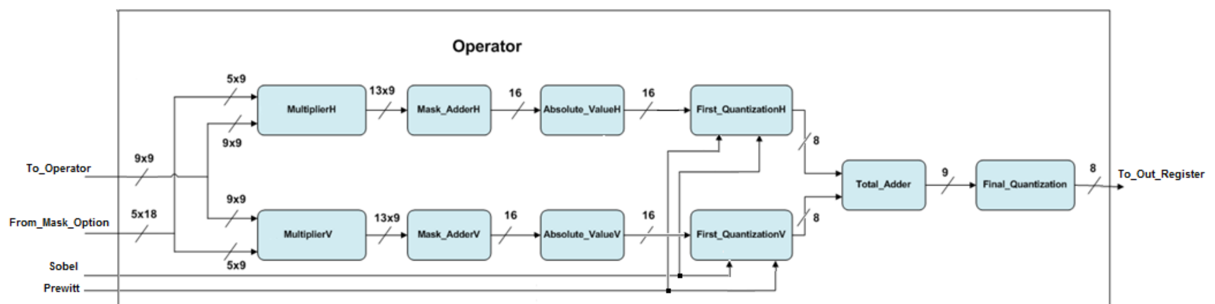


Figura 3.12 Bloque *Operator*.

El bloque Operator es un bloque combinatorio que cuenta con cuatro señales de entrada: From_Mask_Option (5x18) bits y To_Operator (9x9) bits las cuales proporcionan la información de las máscaras -horizontal y vertical- y los valores de la vecindad de píxeles respectivamente y las señales Sobel y Prewitt que especificarán que tipo de *cuantización* se necesita.

MultiplierH y MultiplierV

Cada uno de los nueve píxeles del vecindario que se está operando será multiplicado con su correspondiente coeficiente de la ventana horizontal y vertical, para esta multiplicación se tienen los bloques MultiplierH para la máscara horizontal y MultiplierV para la vertical, la FPGA Cyclone II EP2C20 tiene 26 multiplicadores embebidos, de los cuales se usan 18 para las multiplicaciones de los dos bloques. El peor caso será un píxel cuyo valor de intensidad sea de 255 y sea multiplicado por un coeficiente de 9 o -9, en cuyo caso la salida será de ±2295 para poder representar este valor, la salida de estos bloques es de (9x13)bits y está conectada al siguiente sub bloque Mask_Adder.

Mask_AdderH y Mask_AdderV

Provenientes de los bloques de multiplicación, llegan a los bloques Mask_Adder nueve valores de un ancho de 13bits, los cuales serán sumados con una disposición de Árbol de Wallace. Este tipo de configuración se explica a continuación:

Árbol De Wallace

Para casos en que se desea sumar más de dos números, se necesita esperar el *Carry* de la operación anterior para continuar con la suma. Esto requiere un total de $m-1$ adiciones, para un retardo en las compuertas de un total de $m \times \log(n)$ el tiempo que tarde en pasar a través de un elemento lógico (t_{LE}), esto suponiendo que se implementa un *Look Ahead Adder* o *Full Adder*. Un *árbol de Wallace* puede realizar la misma operación en tan solo $\log(m) \times \log(n)\, t_{LE}$. El método que realiza el árbol de Wallace consiste en tomar tres números, sean $X + Y + Z$ y convertirlo en dos números $C + S$ tal que $X + Y + Z = C + S$ en un t_{LE}. La razón por la cual un Full Adder no puede realizarlo en tan poco tiempo, como se mencionó anteriormente, radica en la espera que se debe hacer para obtener el carry de la operación preliminar. El *árbol de Wallace* implementa *Carry Save Addition*. Con el cual se evita la espera generada por el *Carry* hasta que el paso final, correspondiente a la suma mediante un *Full Adder* . Con los *Carry Save adders*, el procedimiento se parte en dos partes: la primera calcula la suma entre tres números ignorando cualquier *Carry* generado y la segunda calcula el *Carry* independientemente del primer procedimiento. Luego realiza la suma entre los resultados obtenidos por la primera y segunda parte independientemente, como los muestra la Figura 3.13. Al final del procedimiento, sin embargo es necesario un *Full Adder* para obtener el resultado total. Esto logra el

objetivo de convertir tres números que se desean sumar en dos números, que sumados obtienen el resultado en un tiempo t_{LE}.

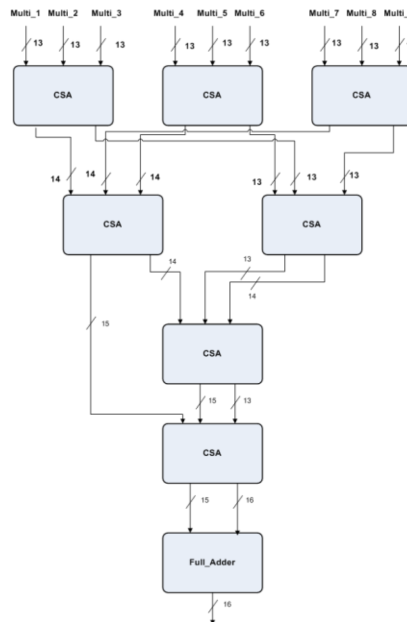


Figura 3.133 *Árbol de Wallace* de nueve entradas de 13 Bits.

La estructura de *Carry Save Adder* corresponde a la misma implementada en un *Full Adder* pero con algunas señales renombradas como lo muestra la Figura 3.14. La entrada C_{in} es renombrada como Z_i y la salida C_{out} como C_i. Tenga en cuenta que debido al desplazamiento del *Carry*, C_0 debe ser sumado con S_1 (S_0 debe permanecer en cero).

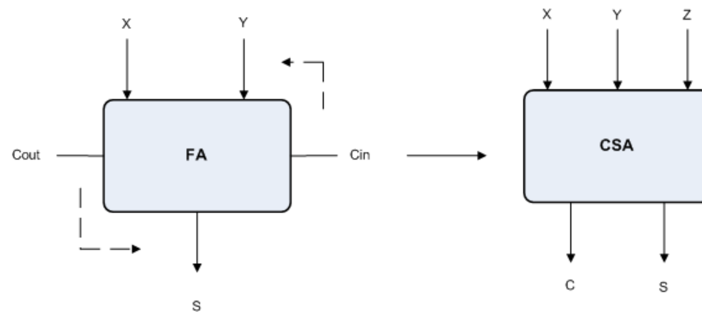


Figura 3.14 Conversion de *Full Adder* a *Carry Save Adder*.

La salida de los bloques Mask_Adder será de 16 bits que equivalen a la suma del peor de los casos del bloque de multiplicación, esta salida entrará al siguiente sub bloque denominado Absolute_Value.

Absolute_ValueH y Absolute ValueV

Después de obtener el resultado de la suma de las multiplicaciones entre los píxeles y los coeficientes de la ventana de filtrado se procede a encontrar el valor absoluto usando el operador ABS encontrado en la biblioteca de funciones aritméticas ieee.numeric_std.

First_QuantizationH y First_QuantizationV

Habiendo calculado el valor absoluto de la suma, es necesario hacer una primera cuantización para volver a un valor entre 0-255 (8bits), esta cuantización depende de los coeficientes usados en la máscara del filtro, la arquitectura propuesta implementa únicamente los algoritmos de Sobel y Prewitt, debido a esto, los factores por los cuales el sistema cuantizará son 4 y 3 respectivamente, las señales de entrada Sobel y Prewitt -activas en alto- habilitarán la correspondiente cuantización.

Total_Adder

Las salidas de First_QuantizationH y First_QuantizationV son de 8 bits que a su vez son las entradas para el bloque Total_Adder, el cual sumará las operaciones hechas por la ventana horizontal con las realizadas por la ventana vertical, para esta tarea se implementó un Full Adder con una salida de 9bits que conecta con el último sub bloque de la sección de Procesamiento de datos.

Final_Quantization

La cuantización final es el último segmento de esta etapa, esta cuantización se hará por 2, ya que en el sub bloque anterior se sumaron dos valores de 8 bits con un resultado de 9bits, la salida del cuantizador final como es de esperarse es de 8bits y es el valor que tomará la imagen resultante en el píxel que se filtro, esta salida llamada To_Out_Register se conecta con el bloque Out_Register, que se examina más adelante.

3.2.3 Unidad de control

Esta unidad es la encargada de administrar el correcto funcionamiento del sistema, sus salidas controlan el resto del sistema, suministra las señales de control a las secciones de adquisición de datos y Procesamiento de datos al igual que es la unidad encargada de habilitar la memoria externa SRAM para su lectura o escritura según sea el caso. Para su inicialización y funcionamiento la unidad de control tiene dos entradas externas al sistema que son: CLK y Reset.

Las señales de control proporcionadas por la unidad de control y su respectivo ancho de bus se muestran en la siguiente tabla 3.2:

SALIDA	ANCHO DE BUS (BITS)
Embedded_Address_0	8
Embedded_Address_1	8
Embedded_Address_2	8
Embedded_WE	9
Embedded_Mux_Control	1
Control_Register_Decoder	1
Mask_Select	9
OutRegister_En	1
Byte_Selector	1
Bidir_OE	1
SRAM_Control	5
Address_SRAM	18
DONE	1

Tabla 3.2 Señales de la Unidad de Control

Esta unidad de control se sub divide en 4 bloques conectados a un bloque principal –Control Unit- los cuales interactúan para suministrar las direcciones y habilitaciones que se necesitan para el correcto filtrado de la imagen.

Control_Unit

Se denomina Control_Unit al bloque principal de la Unidad de Control, este bloque consta de una máquina de estados FSM (*finite state machine*) de 240 pasos, divididos en 4 periodos o fases, el primer periodo (pasos 0-39) se encarga de cargar en las memorias RAM embebidas las tres primeras líneas de la imagen a filtrar, la segunda fase (pasos 40-102), comienza a filtrar la segunda fila de la imagen y por cada píxel filtrado y guardado en la memoria SRAM, se carga un píxel de la imagen original que será usado en un momento posterior, así pues al filtrar la segunda fila completa de la imagen, también se habrá cargado la información de la cuarta fila de píxeles a un modulo 1 del bloque Embedded, la tercera fase de la máquina de estados (pasos 103-169), filtrara la tercera fila de la imagen y cargara la quinta fila en el modulo 2, análogamente la cuarta etapa de la máquina de estados (pasos 170-240) filtrará la cuarta fila y cargará la sexta fila al modulo 3, al completarse la cuarta fase, el sistema pasara a la segunda fase que esta vez no filtrara la segunda fila sino la quinta fila que se encontrará almacenada en el modulo 2. Sucesivamente la máquina de estados pasara por las fases 2, 3 y 4 hasta que filtre la fila 479 y en ese momento romperá su ciclo para activar la señal de salida DONE indicando la terminación del proceso.

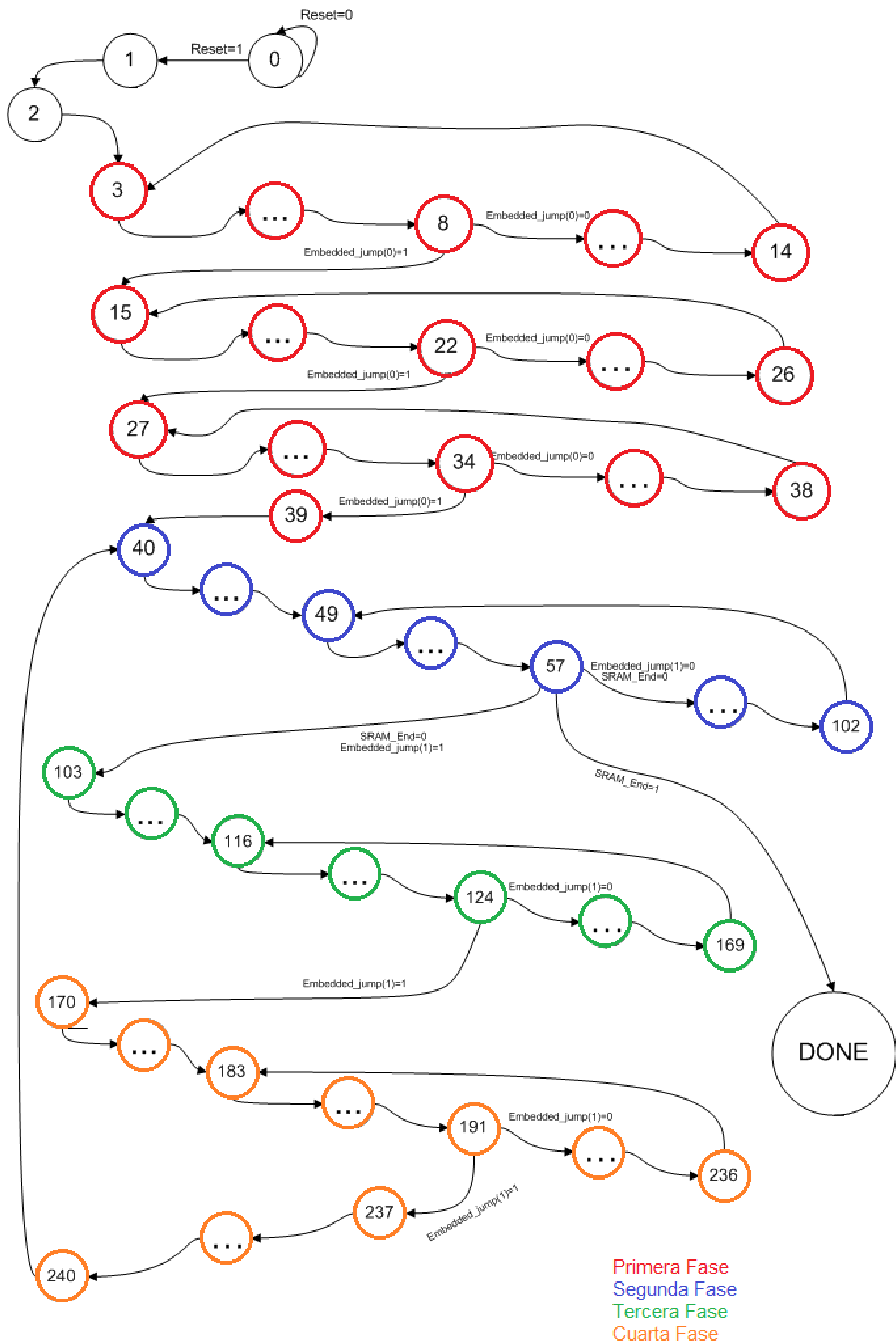


Figura 3.15 Máquina de estados.

A partir de la Figura 3.15 se ve la interacción de la máquina de estados con los demás bloques de la Unidad de Control, la señal Embedded_Jump proveniente del bloque Embedded_Address dictará los parámetros de avance de la máquina de estados, por otra parte la señal SRAM_End generada por el bloque SRAM_Addres_Read le indicará al sistema cuando ha terminado de filtrar la imagen y este procederá a activar la señal de salida DONE.

A continuación se encuentra una lista de las señales de salida del bloque Control_Unit y su función:

SALIDA	FUNCIÓN
Addres_Embedded_Control (0)	Da la orden de incrementar +1 la dirección de escritura de las memorias RAM embebidas.
Addres_Embedded_Control (1)	Incrementa +1 la dirección de lectura de las memorias RAM embebidas 1, 4 y 7.
Addres_Embedded_Control (2)	Incrementa +1 la dirección de lectura de las memorias RAM embebidas 2, 5 y 8.
Addres_Embedded_Control (3)	Incrementa +1 la dirección de lectura de las memorias RAM embebidas 3, 6 y 9.
Addres_Embedded_Control (4)	Controla que tipo de dirección le llega a las nueve memorias RAM embebidas, ya sea la dirección de lectura o la de escritura.
Embedded_WE (9bits)	Habilita la escritura de las memorias RAM embebidas, siendo la posición 0 para la RAM 0, la posicion1 para la RAM 1 y así hasta la posición 9 para la RAM 9.
Embedded_Mux_Control (1bit)	Seleciona el byte que será escrito en la memoria RAM embebida. 0 – Upper Byte 1 – Lower Byte
Control_Register_Decoder (1bit)	Habilita la grabación de los coeficientes de las máscaras horizontal y vertical del filtro a implementar.
Mask_Select (9bits)	Entre las nueve posibles rotaciones de máscaras, escogerá la apropiada para la operación en un momento dado.
OutRegister_En (1bit)	Habilita la escritura del resultado de Operator en un registro para su posterior escritura en la memoria SRAM.
Byte_Selector (1bit)	Le indica al bloque Out_Register que tipo de byte está escribiendo: 0 – Upper Byte 1 – Lower Byte

Bidir_OE (1bit)	Habilita el bloque Bidir para recibir o enviar datos desde o hacia la memoria SRAM 0 – Recibe 1 – Envía
SRAM_Control (5bits)	Maneja las 5 señales de control de la memoria SRAM, las señales CE y OE están pegadas a bajo (Low).
SRAM_Address_Read_Control	Incrementa +1 la dirección de lectura de la memoria SRAM.
SRAM_Address_Write_Control	Incrementa +1 la dirección de escritura de la memoria SRAM.
SRAM_Address_ Control	Controla que tipo de dirección le llega a la memoria SRAM, ya sea la dirección de lectura o la de escritura.
DONE	Indica que el filtrado ha finalizado y se prende el LEDG0 de la tarjeta de desarrollo.

Embedded_Address

Este bloque es el encargado de proveer las direcciones de lectura y escritura para las nueve memorias RAM embebidas, su estructura interna está compuesta por cuatro contadores de los cuales uno maneja la dirección de escritura y los otros tres proporcionan las tres distintas direcciones de lectura, una dirección de lectura para cada posición de los módulos del bloque **Embedded**, estas direcciones serán administradas por medio de la señal de control Address_Embedded_Control la cual también controlará un multiplexor encargado de enrutar las direcciones de lectura o escritura por las tres salidas del bloque, estas son: Embedded_Addres_0, Embedded_Addres_1 y Embedded_Addres_2.

La dirección de escritura se envía igual por las tres salidas ya que esta dirección será usada por solo una memoria embebida a la vez, llegando a un valor máximo de 213 con lo que se garantiza recorrer todas las posiciones de las memorias RAM embebidas, para las direcciones de lectura se envían distintas direcciones por cada una de las salidas como se ve en la figura 3.16, debido a que la lectura de las memorias se hace en forma paralela y no necesariamente se leen las mismas posiciones en las memorias RAM embebidas.

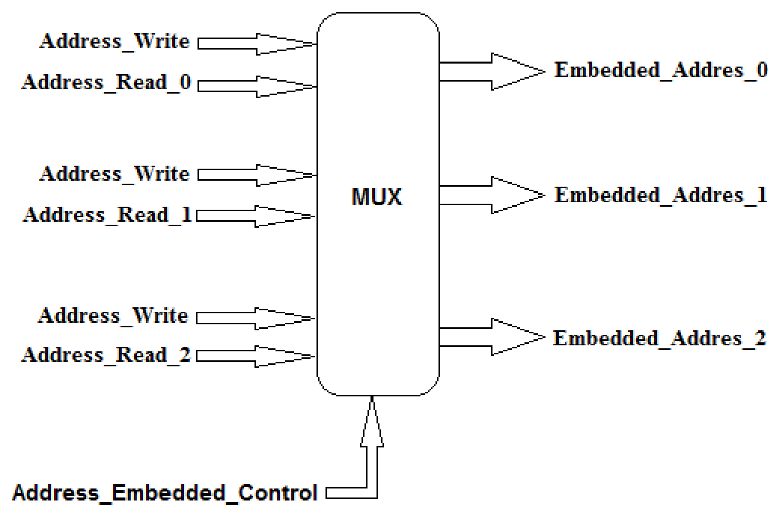


Figura 3.16 Multiplexor de bloque *Embedded_Address.*

Para indicar al bloque **Control_Unit** que el sistema terminó de cargar una fila de píxeles, **Embedded_Address** usa la salida Embedded_Jump en la cual con un alto le indica al control que la dirección de escritura llego a 213, con lo cual se da por entendido que terminó de recorrer todas las posiciones de memoria de un modulo en el bloque **Embedded**.

SRAM_Address_Read, SRAM_Address_Write y SRAM_Address

Estos tres bloques tienen una función análoga al bloque **Embedded_Address**, los dos primeros bloques son contadores que entregarán la dirección de lectura y escritura respectivamente al bloque SRAM_Address el cual actuará como multiplexor dejando pasar la dirección requerida según lo indique la señal de control SRAM_Address_Control, con un bajo para dejar pasar la dirección de lectura y un alto para la dirección de escritura, la dirección final será conectada a la señal de salida Address_SRAM.

Recordando que los píxeles del borde de la imagen no se filtrarán, **SRAM_Address_Write** comenzará en 320, que es la dirección del primer píxel que será filtrado, aumentará +1 la dirección con cada alto en la señal SRAM_Address_Write_Control llegando hasta la dirección 153279 que corresponde al último píxel filtrado, el bloque **SRAM_Address_Read** comienza en la dirección 0 e irá aumentando con un alto en la señal SRAM_Address_Read_Control hasta la dirección 153599 con lo que habrá recorrido toda la matriz de la imagen, el bloque **Control_Unit** se retroalimenta del bloque **SRAM_Address_Read** mediante la señal SRAM_End que se pone en alto una vez se haya terminado de cargar la última fila de la imagen en la memoria RAM embebida y así el sistema sabrá que solo tiene que filtrar una fila mas y acabará el proceso una vez esta sea filtrada.

3.2.4 Etapa de salida

La etapa de salida del sistema está conformada por un registro llamado **Out_Register** y un bloque bidireccional denominado **Bidir**, el resultado de las operaciones realizadas por el bloque **Operator** son almacenadas en el registro de salida **Out_Register** siendo este habilitado por la señal de control OutRegister_En, para luego ser enviado como lower o upper byte -según lo indique la señal de control Byte_Selector- al bloque **Bidir** el cual con un arreglo de buffers triestado logra enviar y recibir la información por los pines bidireccionales del chip FPGA según el valor de la señal de control Bidir_OE.

3.3 DIAGRAMA DE TIEMPOS

Consultar Anexo Digital

3.4 DESCRIPCIÓN AHPL

Consultar Anexo Digital

3.5 DESCRIPCIÓN VHDL

Consultar Anexo Digital

4. ANÁLISIS DE RESULTADOS

La implementación de la arquitectura realizada siguiendo los parámetros sugeridos por la sección de técnicas digitales para este trabajo de grado, dio como resultado una máquina de estados con un total de doscientos cuarenta estados en codificación *One Hot*, cuya función es controlar los veinte bloques necesarios para el procesamiento de imágenes sobre la FPGA. Como se mencionó anteriormente, la totalidad de esta arquitectura es realizada en VHDL e implementado sobre una tarjeta de desarrollo ALTERA Cyclone II EP2C20F484C7 cuyas simulaciones y resultados se obtuvieron bajo el software QUARTUS II de altera diseñado para programar tarjetas de desarrollo como la que fue implementada para este trabajo de grado.

4.1 REPORTE GENERADO POR QUARTUS II DE ALTERA®

4.1.1 Recursos empleados por el dispositivo programable

Flow Status	Successful - Tue May 17 14:30:16 2011
Quartus II Version	9.1 Build 350 03/24/2010 SP 2 SJ Web Edition
Revision Name	filter
Top-level Entity Name	filter
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Met timing requirements	No
Total logic elements	634 / 18,752 (3 %)
Total combinational functions	634 / 18,752 (3 %)
Dedicated logic registers	320 / 18,752 (2 %)
Total registers	320
Total pins	189 / 315 (60 %)
Total virtual pins	0
Total memory bits	18,432 / 239,616 (8 %)
Embedded Multiplier 9-bit elements	18 / 52 (35 %)
Total PLLs	0 / 4 (0 %)

Figura 4.1 Reporte recursos utilizados generado en QUARTUS II®.

En el reporte generado por QUARTUS II (Figura 4.1), se destacan los dieciocho multiplicadores correspondientes a los nueve multiplicadores en el bloque Horizontal_MultiplierH y nueve del bloque Vertical_MultiplierV contenidos en el bloque Operator. Así mismo se puede observar un 8% empleado de la memoria embebida, es decir 18,432 bits. Estos bits se encuentran empleados en el bloque E**mbedded** cuyo propósito es almacenar los valores de los píxeles a los que se les deben aplicar la máscara. Este resultado proviene de las nueve memorias de doscientas cincuenta y seis palabras de ocho bits de ancho, necesarias para almacenar las tres filas que se les desea aplicar el filtrado. El resultado es $Memory\ Bits = 9memorias \times 256posiciones \times 8bits = 18{,}432bits$. Los elementos restantes son empleados en los bloques adicionales, operadores lógicos, maquina de estados e interfaces utilizadas para la arquitectura de este diseño digital.

4.1.2 Análisis de tiempos de propagación

Type	Message
	Info: Quartus II Assembler was successful. 0 errors, 0 warnings
	Info: **
	Info: Running Quartus II Classic Timing Analyzer
	Info: Command: quartus_tan --read_settings_files=off --write_settings_files=off filter -c filter --timing_analysis_only
	Info: Longest tpd from source pin "Reg1_4[0]" to destination pin "out_quantization[7]" is 68.327 ns
	Info: Quartus II Classic Timing Analyzer was successful. 0 errors, 0 warnings
	Info: Quartus II Full Compilation was successful. 0 errors, 10 warnings

Figura 4.2 Reporte de retardos de bloque *Operator* en QUARTUS II.

El reporte generado por QUARTUS II que muestra la Figura 4.2 muestra el mayor retardo que el bloque **Operator** puede generar. Este bloque presenta el mayor retardo en el sistema debido a que la arquitectura de este sistema está compuesta en su mayoría por redes combinatorias generadas por sus elementos lógicos (LE), como se aprecia en la Figura 4.3, 634 LE y 18 multiplicadores embebidos.

La minimización de tiempo para este bloque es crítica, ya que con operadores convencionales como por ejemplo un *full adder*, su tiempo de propagación es mucho mayor que el presentado por el árbol de Wallace. Estos retardos se verían reflejados en valores erróneos durante una operación con un reloj de alta velocidad. Para el bloque combinatorio **Operator** se destinaron cuatro ciclos de reloj antes de almacenar el valor en los registros que se localizan en el bloque Out_Register. Implementada con un reloj de 50Mhz (periodo de 20ns), la operación matemática tendrá una ventana de tiempo de 80ns, esta operación según el informe presentado por QUARTUS II (Figura 4.2) tendrá un retardo máximo de 68.327ns con lo cual se garantiza el correcto funcionamiento de este bloque.

Flow Status	Successful - Tue May 17 17:51:18 2011
Quartus II Version	9.1 Build 350 03/24/2010 SP 2 SJ Web Edition
Revision Name	filter
Top-level Entity Name	filter
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Met timing requirements	Yes
Total logic elements	634 / 18,752 (3 %)
Total combinational functions	634 / 18,752 (3 %)
Dedicated logic registers	0 / 18,752 (0 %)
Total registers	0
Total pins	211 / 315 (67 %)
Total virtual pins	0
Total memory bits	0 / 239,616 (0 %)
Embedded Multiplier 9-bit elements	18 / 52 (35 %)
Total PLLs	0 / 4 (0 %)

Figura 4.3 Recursos utilizados bloque *Operator* por QUARTUS II.

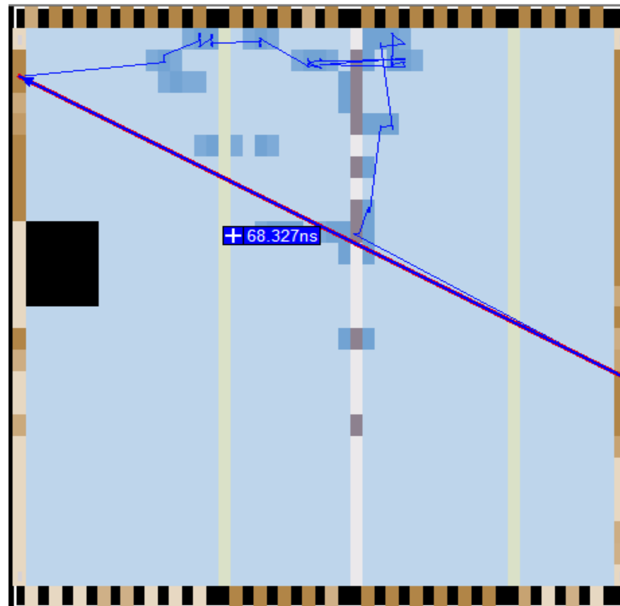


Figura 4.4 *Floorplan* generado por QUARTUS II.

El *Floorplan* presentado en la figura 4.4 muestra el camino más largo que existe en el bloque **Operator**, este es generado desde el bit cero de la memoria embebida cuatro, hasta el bit de salida siete del bloque **Final_Quantization**. Este resultado hace parte de la compilación generada por QUARTUS II que no permite una modificación manual o cambio alguna sobre esta. Para el caso en que se desee disminuir el tiempo de propagación de este bloque, será necesario realizar bloques combinatorios más rápidos, pero para propósitos de este trabajo de grado cumple con los requerimientos mínimos para un reloj de 50Mhz.

4.1.3 Asignación de pines tarjeta de desarrollo

Clk	Input	PIN_D12	3	B3_N0
Done	Output	PIN_U22	6	B6_N1
prewitt	Input	PIN_M1	1	B1_N0
Reset	Input	PIN_T21	6	B6_N0
Sobel	Input	PIN_L2	2	B2_N1
SRAM_ADDRESS_OUT[17]	Output	PIN_Y5	8	B8_N1
SRAM_ADDRESS_OUT[16]	Output	PIN_Y6	8	B8_N1
SRAM_ADDRESS_OUT[15]	Output	PIN_T7	8	B8_N1
SRAM_ADDRESS_OUT[14]	Output	PIN_R10	8	B8_N0
SRAM_ADDRESS_OUT[13]	Output	PIN_U10	8	B8_N0
SRAM_ADDRESS_OUT[12]	Output	PIN_Y10	8	B8_N0
SRAM_ADDRESS_OUT[11]	Output	PIN_T11	8	B8_N0
SRAM_ADDRESS_OUT[10]	Output	PIN_R11	8	B8_N0
SRAM_ADDRESS_OUT[9]	Output	PIN_W11	8	B8_N0
SRAM_ADDRESS_OUT[8]	Output	PIN_V11	8	B8_N0
SRAM_ADDRESS_OUT[7]	Output	PIN_AB11	8	B8_N0
SRAM_ADDRESS_OUT[6]	Output	PIN_AA11	8	B8_N0
SRAM_ADDRESS_OUT[5]	Output	PIN_AB10	8	B8_N0
SRAM_ADDRESS_OUT[4]	Output	PIN_AA5	8	B8_N1
SRAM_ADDRESS_OUT[3]	Output	PIN_AB4	8	B8_N1
SRAM_ADDRESS_OUT[2]	Output	PIN_AA4	8	B8_N1
SRAM_ADDRESS_OUT[1]	Output	PIN_AB3	8	B8_N1
SRAM_ADDRESS_OUT[0]	Output	PIN_AA3	8	B8_N1
SRAM_CE	Output	PIN_AB5	8	B8_N1
SRAM_DATA[15]	Bidir	PIN_U8	8	B8_N1
SRAM_DATA[14]	Bidir	PIN_V8	8	B8_N1
SRAM_DATA[13]	Bidir	PIN_W8	8	B8_N1
SRAM_DATA[12]	Bidir	PIN_R9	8	B8_N0
SRAM_DATA[11]	Bidir	PIN_U9	8	B8_N0
SRAM_DATA[10]	Bidir	PIN_V9	8	B8_N1
SRAM_DATA[9]	Bidir	PIN_W9	8	B8_N0
SRAM_DATA[8]	Bidir	PIN_Y9	8	B8_N0
SRAM_DATA[7]	Bidir	PIN_AB9	8	B8_N0
SRAM_DATA[6]	Bidir	PIN_AA9	8	B8_N0
SRAM_DATA[5]	Bidir	PIN_AB8	8	B8_N0
SRAM_DATA[4]	Bidir	PIN_AA8	8	B8_N0
SRAM_DATA[3]	Bidir	PIN_AB7	8	B8_N1
SRAM_DATA[2]	Bidir	PIN_AA7	8	B8_N1
SRAM_DATA[1]	Bidir	PIN_AB6	8	B8_N1
SRAM_DATA[0]	Bidir	PIN_AA6	8	B8_N1
SRAM_LB	Output	PIN_Y7	8	B8_N1
SRAM_OE	Output	PIN_T8	8	B8_N1
SRAM_UB	Output	PIN_W7	8	B8_N1
SRAM_WE	Output	PIN_AA10	8	B8_N0

Figura 4.5 Asignación de pines de tarjeta de desarrollo.

Como se explicó en el Capitulo 3, el sistema está compuesto por las entradas externas Prewitt, Reset y Sobel, así como la entrada interna CLK, las cuales se puede apreciar en la primera parte de la figura 4.5. Hay que tener en cuenta que esta asignación de pines corresponde únicamente a la tarjeta de desarrollo Cyclone II EP2C20F484C7 de Altera y si se desea configurar en cualquier otra tarjeta de desarrollo, debe tener como requisitos mínimos un bus de datos entre Memoria externa y FPGA de 16bits con una capacidad de direccionamiento de al menos 18bits además de disponer de tres botones o interruptores.

4.1.4 Diagrama De Tiempos

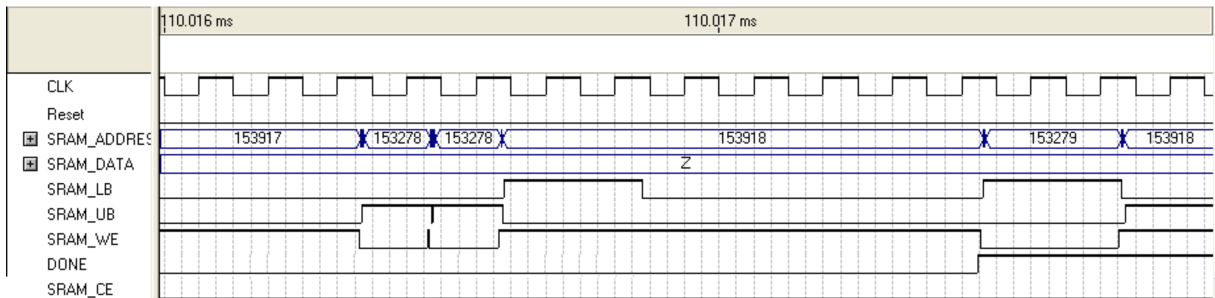


Figura 4.6 Diagrama de tiempos fin de procesamiento 27Mhz.

El diagrama de tiempos generado por Quartus II (Figura 4.6) muestra la habilitación de escritura para la dirección 153279 de la memoria SRAM, donde se encuentra el pixel 306558 que corresponde al último pixel que se debe operar ubicado en la posición (639,479). El fin de este procedimiento se refleja en la señal DONE que es asignada a un *led* de la tarjeta de desarrollo indicándole al usuario el fin del procedimiento. Para esta simulación se utilizó un reloj de 27MHZ, dando por finalizado el procesamiento de la imagen en exactamente 110.016ms.

Inicialmente se pensó en implementar el sistema bajo el reloj de 27Mhz debido a los retardos de las compuertas en el bloque Operator, sin embargo este bloque fue reformado con operadores lógicos más eficientes, con lo cual fue posible implementar el sistema con un reloj de 50Mhz. Con la modificación de este bloque se obtiene un tiempo de procesamiento de 55ms, como se muestra en la Figura 4.7. Vale la pena aclarar, que modificación del código se refiere al cambio de sumadores convencionales *Full Adder* a sumadores más eficientes como los son los Arboles de Wallace. Además de esto se implementó la librería estándar para VHDL IEEE.NUMERIC_STD.ALL, que contiene las operaciones lógicas para VHDL de valor absoluto y división.

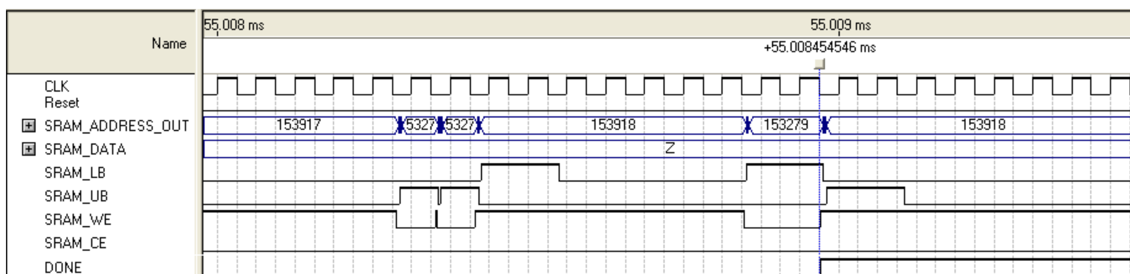


Figura 4.7 Diagrama de tiempos fin de procesamiento 50Mhz.

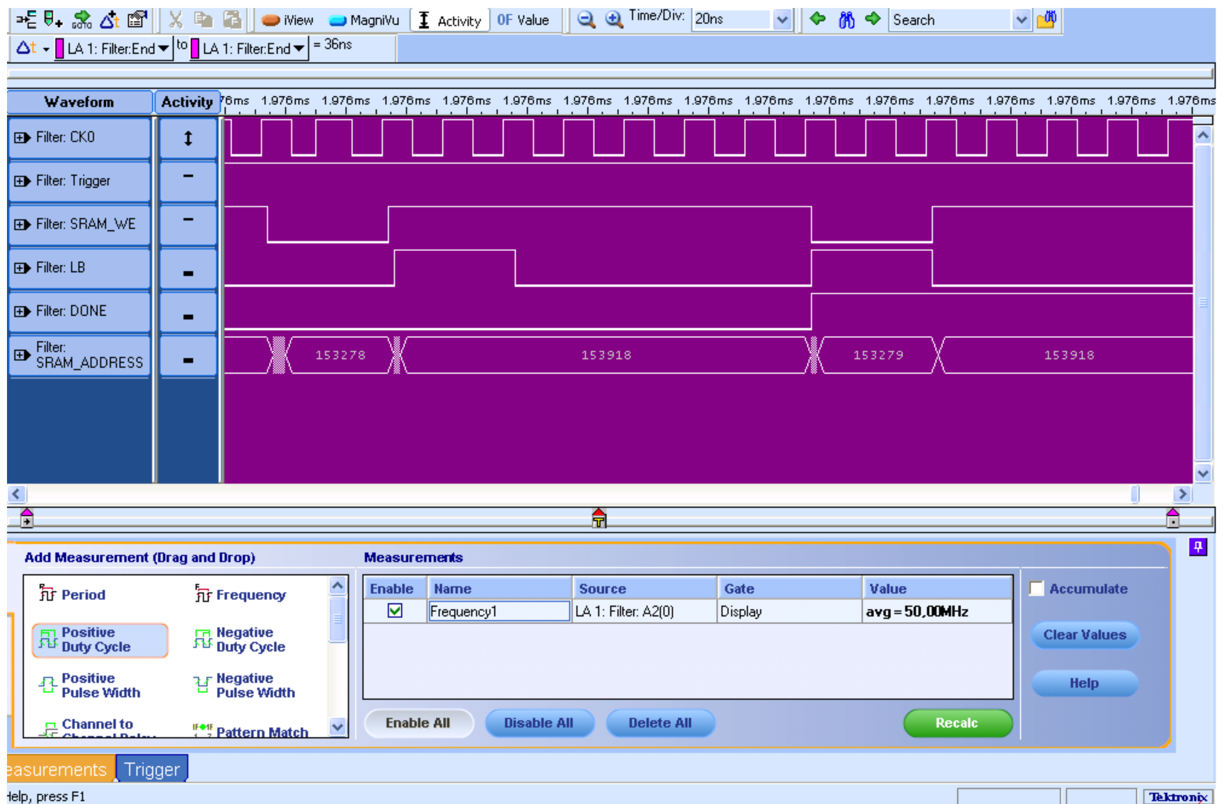


Figura 4.8 Diagrama de tiempos analizador lógico TLA52202B®.

La grafica de la Figura 4.8 muestra el mismo resultado obtenido a nivel práctico dado por el analizador lógico ratificando como la señal DONE se activa en el momento que el ultimo pixel que se debe procesar, almacena su valor en la dirección 153279 de la memoria SRAM correspondiente al último pixel filtrado (639,479) de la imagen.

4.2 PROTOCOLO DE PRUEBAS

4.2.1 Imágenes de prueba

El resultado del procesamiento de filtrado de una imagen, implementado en FPGA fue analizado y corroborado bajo la implementación de los algoritmo Sobel y Prewitt adaptados a lenguaje de programación de medio nivel C y lenguaje de alto nivel como lo es MATLAB®. Los códigos de C y de MATLAB se encuentran en el Anexo B y Anexo A correspondientemente. Los resultados que se obtuvieron en los tres casos comprueban el éxito del algoritmo implementado en FPGA obteniendo los mismos valores en los tres casos. De las diversas pruebas que se realizaron con distintas imágenes se aprecia en la Figura 4.9, una imagen en escala de grises a la que se le aplicó el filtro.

Figura 4.9 Imagen de prueba en escala de grises. [3]

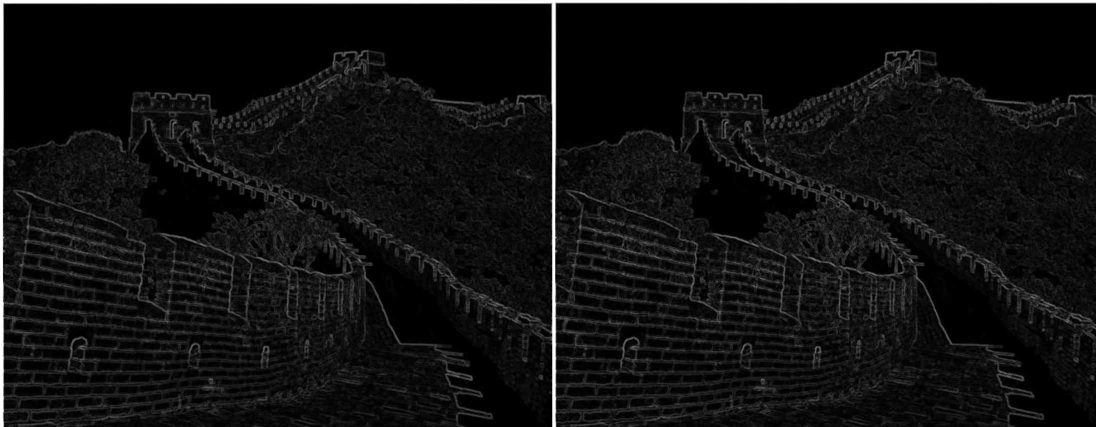


Figura 4.10b Imagen resultante filtro Sobel Figura 4.10a Imagen resultante filtro Prewitt.

Al aplicar el filtro Sobel y Prewitt a la imagen de prueba en FPGA se obtuvo el resultado en 55ms este se puede apreciar en la Figura 4.10. En esta figura se observa que efectivamente los algoritmos implementados sobre la imagen de prueba identifican los bordes o discontinuidades significativas presentadas en la imagen, en la mayoría de los casos no es adecuado realizar dichas deducciones de una manera subjetiva, por este motivo se recurrió un análisis más riguroso apoyado en *software* de hojas de cálculo, para así obtener un resultado más exacto y objetivo en comparación al obtenido por medio del análisis grafico. A continuación los resultados obtenidos por los algoritmos originados en los tres tipos de lenguaje:

	FPGA	MATLAB	C
Total suma pixeles imagen original	36444107		
Total suma pixeles Algoritmo Sobel	4714085	4714085	471085
Total suma pixeles Algoritmo Prewitt	4580675	4580675	4580675
Diferencia pixeles Otros lenguajes	0	0	0

Tabla 4.2 Imagen de prueba en escala de grises

De la Tabla 4.2 se puede deducir, que el algoritmo Sobel presenta mayores valores en los píxeles debido a la diferencia de pesos en los coeficientes, frente al filtro Prewitt que presenta un peso de uno

en todos los coeficientes de la máscara implementada para dicho algoritmo (Figura 1.5 y Figura 1.6). Por esta razón se percibe en la Figura 4.10b una mayor intensidad en los bordes de la imagen resultante.

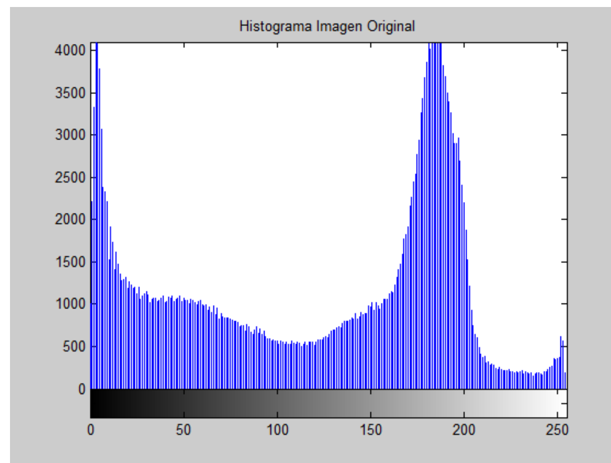


Figura 4.11 Histograma imagen original.

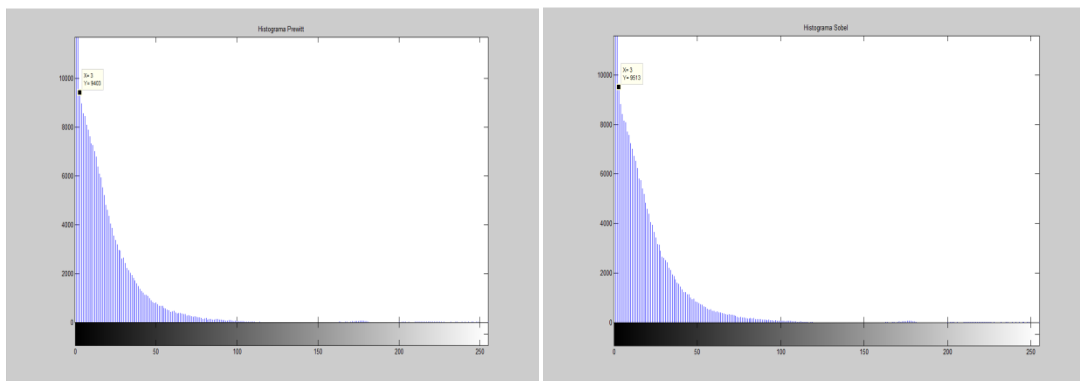


Figura 4.12a Histograma filtro *Prewitt*. Figura 4.12b Histograma filtro *Sobel.*

Con los histogramas de la Figura 4.12 también fue comprobado el resultado exitoso de los dos algoritmos en FPGA, obteniendo el mismo histograma para los resultados del filtrado por los algoritmos en lenguaje de programación C. Con los histogramas presentados en las Figuras 4.11 y 4.12 se observa, la reducción en la cantidad de datos en una grafica para destacar información importante de esta, para este caso los bordes.

4.2.2 Velocidad de procesamiento

En la sección 4.1.4, se encontró que el tiempo para procesar la imagen en la FPGA es de 55ms trabajando con un reloj de 50Mhz, el diseño de la arquitectura para la FPGA fue implementado en VHDL, los resultados fueron analizados, implementando los mismos algoritmos de detección de bordes en lenguaje de programación C y lenguaje de alto nivel en MATLAB y comparando estos tres resultados. MATLAB y C fueron implementados en diferentes procesadores con el fin de tener una

mejor estimación del tiempo que se requiere para terminar procedimiento de detección de bordes con los algoritmos de Sobel y Prewitt, en un computador convencional.

	Computador 1	Computador 2	Computador 3	Computador 4	FPGA
Procesador	Intel Core 2 Duo	Intel Pentium	AMD Turion X2	Intel Centrino	
Velocidad De Procesador	3.00 Ghz	1.87 Ghz	1.6 Ghz	1.83 Ghz	50Mhz
Memoria RAM	3.00 GB	2.00 GB	2.00 GB	2 GB	
Tiempo De Procesamiento Matlab (ms)	592.3	1083.6	1248.5	954.2	55
Tiempo De Procesamiento C (ms)	61.3	83.95	166.97	249.76	

Tabla 4.3 Características Computadores Expuestos A Procesamiento De Imágenes

La tabla 4.3 muestra los resultados obtenidos por cuatro diferentes procesadores empleando el código tanto en C como en MATLAB y el mismo procedimiento implementado en el FPGA, se puede observar que en la relación de la Velocidad de Procesamiento con el Tiempo de Procesamiento es más eficiente el FPGA, igualmente este dispositivo tiene el más pequeño de los Tiempos de Procesamiento, menor incluso que el Tiempo de Procesamiento del programa en C implementado en un Intel® Core 2 Duo y casi 11 veces menor al mismo programa en Matlab.

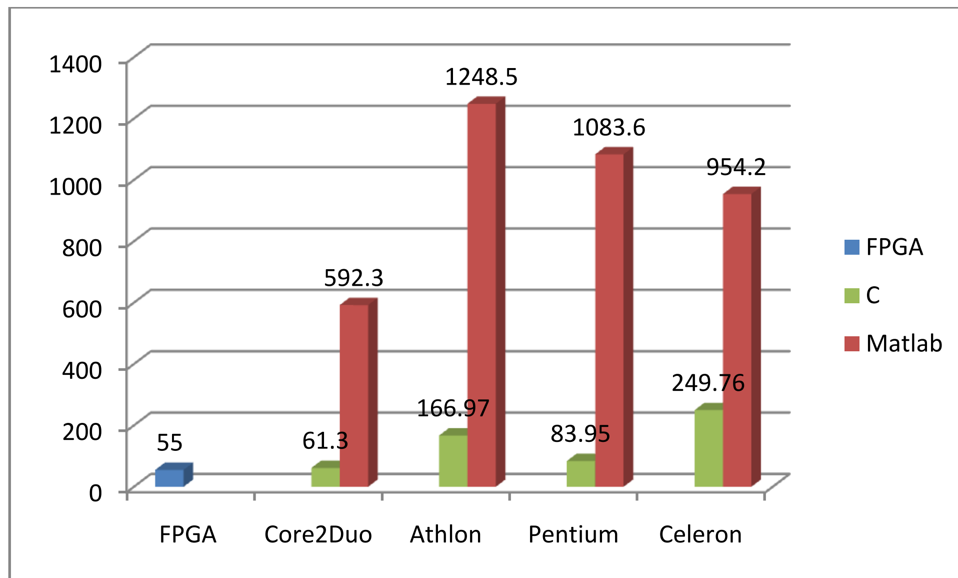


Figura 4.12 Tiempo transcurrido para filtrado de imágenes (ms).

Como se observa en la Figura 4.13 en todos los casos, el código realizado en Matlab presenta mayor latencia frente a los demás lenguajes de programación. Esto debido a que no es un lenguaje de programación en sí, sino que es un lenguaje de programación interpretado que requiere más tiempo de procesamiento frente al mundialmente conocido lenguaje de programación C, esto descarta a Matlab como punto de comparación para la FPGA. En contrapartida se tiene el código realizado en C, donde se tienen dos resultados importantes: el código usado en el procesador Core2Duo que presenta una latencia de 61.3ms y Pentium con 83.95ms de latencia.

La FPGA presenta un procesamiento de la imagen 10.27% más rápido que un computador convencional con un procesador 98.17% más rápido que el reloj del FPGA. Este resultado no pretende dar como conclusión que la FPGA es más veloz para realizar procesamiento de imágenes que un

computador convencional, simplemente pretende explorar formas alternativas en el procesamiento de imágenes dejando a la FPGA como punto de partida para la implementación de algoritmos matemáticos para el procesamiento de imágenes.

4.2.3 Implementación en distintas tarjetas de desarrollo

Flow Status	Successful - Sat May 21 13:48:45 2011
Quartus II Version	9.1 Build 350 03/24/2010 SP 2 SJ Web Edition
Revision Name	filter
Top-level Entity Name	filter
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Met timing requirements	No
Total logic elements	2,004 / 18,752 (11 %)
Total combinational functions	1,884 / 18,752 (10 %)
Dedicated logic registers	320 / 18,752 (2 %)
Total registers	320
Total pins	45 / 315 (14 %)
Total virtual pins	0
Total memory bits	18,432 / 239,616 (8 %)
Embedded Multiplier 9-bit elements	0 / 52 (0 %)
Total PLLs	0 / 4 (0 %)

Figura 4.14a Resultados FPGA Cyclone II®.

Flow Status	Successful - Sat May 21 13:33:29 2011
Quartus II Version	9.1 Build 350 03/24/2010 SP 2 SJ Web Edition
Revision Name	filter
Top-level Entity Name	filter
Family	Cyclone III
Device	EP3C25F324C8
Timing Models	Final
Met timing requirements	N/A
Total logic elements	2,029 / 24,624 (8 %)
Total combinational functions	1,886 / 24,624 (8 %)
Dedicated logic registers	320 / 24,624 (1 %)
Total registers	320
Total pins	45 / 216 (21 %)
Total virtual pins	0
Total memory bits	18,432 / 608,256 (3 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

Figura 4.15b Resultados FPGA Cyclone II®.

Flow Status	Successful - Sat May 21 13:58:28 2011
Quartus II Version	9.1 Build 350 03/24/2010 SP 2 SJ Web Edition
Revision Name	filter
Top-level Entity Name	filter
Family	Stratix II
Device	EP2S15F672C5
Timing Models	Final
Met timing requirements	No
Logic utilization	14 %
Combinational ALUTs	1,517 / 12,480 (12 %)
Dedicated logic registers	323 / 12,480 (3 %)
Total registers	323
Total pins	45 / 367 (12 %)
Total virtual pins	0
Total block memory bits	18,432 / 419,328 (4 %)
DSP block 9-bit elements	0 / 96 (0 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 2 (0 %)

Figura 4.14c Resultados FPGA Stratix II®.

Los recursos utilizados por las distintas tarjetas de desarrollo, ilustrados en la Figura 4.14 dan como resultado un consumo no mayor al 12% de los elementos lógicos y un máximo del 8% de memoria embebida de dichas tarjeta, dejando espacio suficiente en la FPGA para realizar ajustes o adecuaciones en la arquitectura de este trabajo de grado. Tal caso puede ser el procesamiento de una imagen de mayor tamaño, recepción de la imagen por puerto serial o tratamientos adicionales sobre una imagen.

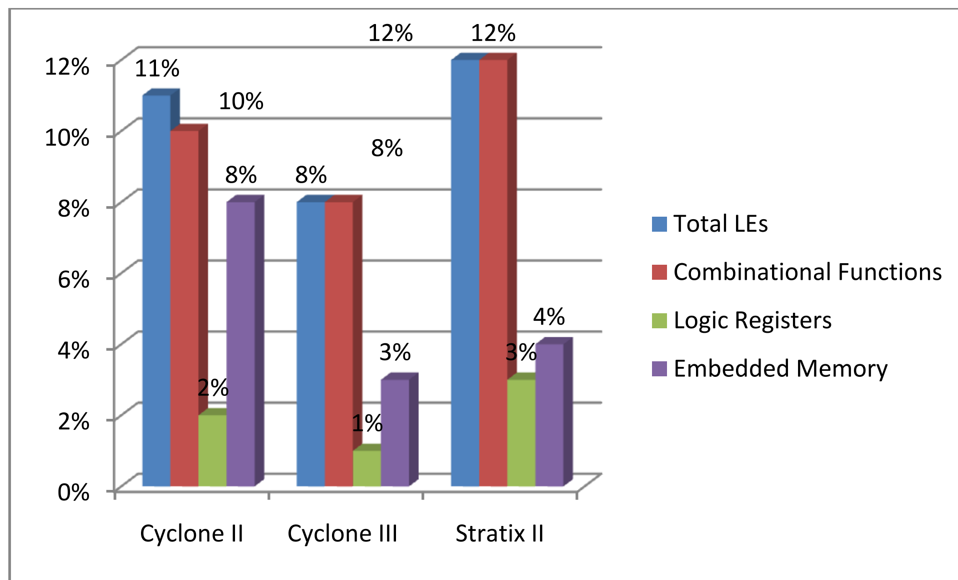


Figura 4.15 Recursos usados por las FPGA (%).

4.2.4 Análisis adicionales

En este trabajo de grado se implementaron imágenes de 640 x 480 correspondiente a un estándar para imágenes para monitores VGA, la limitante inicial era tener una memoria embebida de un tamaño tal que pudiera almacenar las tres filas necesarias para realizar el algoritmo de Sobel Y Prewitt. Si se necesitará implementar este sistema para imágenes de un tamaño mayor como por ejemplo una imagen HD de tamaño 1920 × 1080, bastaría con modificar los contadores contenidos en Embedded_Adress y Address_SRAM y modificar el tamaño de las memorias embebidas, para este ejemplo seria:

$$Memory\ Bits = 3\ filas \times 8\ bits \times 1920\ pixeles = 46{,}800 bit.$$

Este requerimiento de 46,800 bits puede ser cumplido con cualquiera de las tres diferentes tarjetas de desarrollo.

El alcance de este proyecto estaría limitado por el bloque Operator y la velocidad de lectura/escritura de la memoria SRAM implementada, como la memoria presenta un tiempo para lectura de 15ns en el peor caso y se tienen dos ciclos de reloj en la máquina de estados para esta habilitación, no tendría un efecto relevante si se cambiara el reloj del sistema mientras dos periodos de este sean superiores a los 15ns. El tiempo de propagación dado por las operaciones que se deben realizar en el bloque **Operator** que según el reporte generado por Quartus II en la sección 4.1.2, es de 68.327ns. Se podría trabajar a una velocidad máxima de:

$$TiempoDePropagacion = \frac{68.327ns}{4\ periodos} = 17.08ns$$

Es decir,

$$FrecuenciaDeReloj = \frac{1}{17.08ns} = 58.8Mhz$$

Este tiempo de propagación puede ser modificado al hacer un bloque combinatorio más eficiente, evento que no fue de gran importancia para esta arquitectura ya que la velocidad de los relojes de las tres tarjetas de desarrollo era máximo 50Mhz. El mayor tiempo de retardo que se identificó, fue en el bloque **first_Quantization**, este requiere una división y un cambio de señales de *std_logic* a *unsigned*, pero cumple con los requisitos mínimos para una correcta operación.

5. CONCLUSIONES

Con este proyecto fue posible evaluar el potencial de la FPGA para el procesamiento de señales, obteniendo resultados favorables e incluso superiores a los obtenidos en *software*, considerando que este se trabajo en un procesador a una frecuencia la velocidad del procesador del computador con el que se comparó es de más de 2Ghz frente a un FPGA que operó bajo un reloj de tan solo 50Mhz. La ventaja de realizar procesamiento de imágenes sobre una FPGA se debe a la capacidad de realizar procesos con una gran cantidad de datos en paralelo y la posibilidad de realizar una arquitectura dedicada a un proceso especifico.

A pesar de haber implementado el algoritmo de Sobel y Prewitt en la FPGA con un rendimiento adecuado, el tiempo que requirió el diseño y la implementación de la arquitectura en hardware fue considerablemente mayor al de la adaptación del mismo algoritmo en *software*. El tiempo que requiere un diseño de estas características puede reflejarse en perdida de dinero sí el desarrollo de dicho diseño requiere de bloques combinatorios complejos que hicieran el tiempo de procesamiento mayor que el obtenido mediante un *software* convencional. Es necesario evaluar los beneficios de un diseño para *hardware* antes de arrojarse a un diseño digital que en algunos casos podría ser más favorable realizarlo en *software*.

La culminación de este proyecto no hubiera sido posible sin la metodología sugerida en las asignaturas de diseño de sistemas digitales y arquitectura de procesadores, esta técnica fue fundamental para que los resultados obtenidos en el procesamiento fueran los esperados después de compararlos contra el *software*. La metodóloga permitió seguir un orden secuencial con el cual fue posible identificar fácilmente errores en bloques específicos y corregidos durante el diseño, evitando así afectar la totalidad del sistema.

Un aspecto importante a tener en cuenta es el tiempo de propagación generado por el bloque **Operator**, donde se realizaron operaciones como sumas, multiplicaciones, divisiones y valor absoluto. Para el propósito de este proyecto se cumplió con los requisitos mínimos de acuerdo a los tiempos que se asignaron para este bloque de acuerdo a la máquina de estados, si fuera necesario reducir el tiempo de procesamiento bastaría con cambiar este bloque combinatorio por un sistema más eficiente que realice operaciones lógicas a velocidades mayores, como sería el caso de procesamiento de video, que para el caso de 24 fps (*Frames per second*) es suficiente con disminuir el tiempo de procesamiento en 13ns con respecto al obtenido en este proyecto.

La implementación de un *hardware* de propósito específico para el procesamiento de imágenes no ha sido habitual en trabajos de grado realizados en la Pontificia Universidad Javeriana, creemos que hay una falta de interacción para desarrollar proyectos entre las distintas secciones de la facultad. Para el

procesamiento de imágenes, en los últimos años se han desarrollando paquetes de *software* encargados de realizar complejos algoritmos, que facilitan en alguna medida el desarrollo de estos procesos, una desventaja de estos *software* radica en que se verá limitado a la velocidad y nivel de ocupación del procesador en el cual se esté trabajando. La idea de diseñar una arquitectura para implementar estos algoritmos en *hardware* surge del deseo de proponer nuevas alternativas para el procesamiento de imágenes y servir como fundamento para trabajos de grado que continúen con el empeño de unificar estas dos ramas de la electrónica –Técnicas Digitales y Procesamiento de Imágenes- que han dado soluciones a problemas esenciales para el desarrollo de las sociedades modernas.

6. FUENTES DE INFORMACIÓN

[1] ALTERA®. “Cyclone II FPGA Starter Development Board. Reference Manual”, *Internet Archive(www.**altera**.com).*

[2]. BARRETO, Paola M, “Esqueletización de imágenes digitales utilizando F.P.G.A. (Field Programable Gate Array)”, Trabajo de Grado Pontificia Universidad Javeriana, 2006, T.G. 909.

[3] BRION, Shinamoto. “Go Reconfigure.” IEEE Spectrum, *Internet Archive*.

[4] CARDONA, Lina Maria “Funciones De Procesamiento De Imágenes Implementadas En FPGA”, Trabajo de Grado Pontificia Universidad Javeriana, 2009.

[5] DIEDERIK, Verkest. Machine Chameleon.” IEEE Spectrum, *Internet Archive*.

[6] DYLAN McGrath, ”FPGA Market to Pass $2.7 Billion by ’10, In-Stat Says”, EE Times, Mayo 2006. (http://www.eetimes.com/showArticle.jhtml?articleID=188102617)

[7] EFFORD. Nick, “Digital Image Processing a practical introduction using java”, Addison-Wesley.

[8] GONZÁLEZ, Rafael C. “Digital Image Processing”, Prentice-Hall International,pg 136, 2001.

[9] ISSI®. Integrated Silicon Solution, Inc. “Static RAM IS61LV25616 Data Sheet”.

[10] JAIN, Anil K. “Fundamentals of digital image processing”, Prentice-Hall International, pg 349, 1989.

[11] JOHN C. Russ, “The Image Processing Handbook”, Segunda Edicion CRC Press.

[12] MEN Mikro Elektronik GmbH, “Advantage of Field Programmable Gate Arrays”. Manual for FPGA, www.men.de/docs-ext/expertise/pdf/fpga_advantages.pdf, Deutschland, consultado el 3 de febrero de 2011.

[13] PONG P. Chu, “FPGA Prototyping by VHDL examples”, Ed John Wiley & Sons. New jersey USA, 2008.

[14] WEIHUA, Wang, “Reach on Sobel operator for Vehicle Recogniton” IEEE International Joint Conference on Artificial Intelligences, China, 2009.

7. ANEXOS

Anexo A. Evaluación tiempo de procesamiento en MATLAB®

```
% Trabajo De Grado 1032 - Ingeniería Electrónica
% Pontificia Universidad Javeriana
% Alexander Coronado - Luis Andrés Maldonado
% Funcion TimeEval
% Evalua tiempo de procesamiento del Filtro Sobel en MATLAB
close all;
clear all;
gris1 = imread('Original1.bmp');                              % Lee imagen
.bmp en escala de grises del computador
[vertical,horizontal]=size(gris1);                          % Determina tamaño
de la imagen
gris=single(gris1);
t = cputime;                                                % Inicio De Conteo
de procesamiento
for a=2:vertical-1                                          % Recorre la
imagen verticalmente
for b=2:horizontal-1                                        % Recorre la
imagen horizontalmente
% Almacena en mascarax el algoritmo de sobel horizontal.
mascarax(a,b)=gris(a-1,b-1)*(-1)+gris(a-1,b)*(0)+gris(a-
1,b+1)*(1)+gris(a,b-1)*(-2)+gris(a,b)*(0)+gris(a,b+1)*(2)+gris(a+1,b-1)*(-
1)+gris(a+1,b)*(0)+gris(a+1,b+1)*(1);
% Almacena en mascaray el algoritmo de sobel vertical.
mascaray(a,b)=gris(a-1,b-1)*(1)+gris(a-1,b)*(2)+gris(a-1,b+1)*(1)+gris(a,b-
1)*(0)+gris(a,b)*(0)+gris(a,b+1)*(0)+gris(a+1,b-1)*(-1)+gris(a+1,b)*(-
2)+gris(a+1,b+1)*(-1);
end;
end;
total1=abs(floor(mascaray/4))+abs(floor(mascarax/4));     % Suma valor
absoluto de mascarax y valor absoluto de mascaray, y realiza Cuantización
total=(floor(total1/2));                                    % Realiza
Cuantizacion Final
for b=1:horizontal
total(1,b)=gris(1,b);                                       % Agrega primera
fila
total(vertical,b)=gris(vertical,b);                         % Agrega ultima
fila
end;
for b=1:vertical
total(b,1)=gris(b,1);                                       % Agrega primera
columna
total(b,horizontal)=gris(b,horizontal);                     % Agrega ultima
columna
end;
e = cputime-t;                                              % Fin De Conteo de
procesamiento
final=uint8(total);                                         % Almacena el
resultado final
imwrite(final,'ImagenSobel.bmp');                           % Almacena
Resultado
%imshow(final);                                              % Muestra
Resultado Del Algoritmo
```

Anexo B. Evaluación tiempo de procesamiento en C

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
 int imagen; int i=1; int j=1;int a=2;int b=2;
 short image[481][641];
 short mascarax[481][641];
 short mascaray[481][641];
 short total1[481][641];
 clock_t t_ini, t_fin;
 double secs;
main()
{
 FILE *pf;
 FILE *cfPtr;
 if((cfPtr = fopen("imagen.txt", "r")) == NULL)
 printf("No Se Encontro El Archivo\n");
 else   {
 fscanf(cfPtr,"%d", &imagen);
  for(j = 1; j<= 480; j++){
    for(i = 1; i<= 640; i++){
      image[j][i] = imagen;
      fscanf(cfPtr, "%d", &imagen);
                      }
                     }
fclose(cfPtr);
 t_ini = clock();
}
 for(a = 2; a<= 479; a++){
    for(b = 2; b<= 639; b++){
  mascarax[a][b]=floor(abs(image[a-1][b-1]*(1)+image[a-1][b]*(0)+image[a-1][b+1]*(-
1)+image[a][b-1]*(2)+image[a][b]*(0)+image[a][b+1]*(-2)+image[a+1][b-
1]*(1)+image[a+1][b]*(0)+image[a+1][b+1]*(-1))/4);
  mascaray[a][b]=floor(abs(image[a-1][b-1]*(1)+image[a-1][b]*(2)+image[a-
1][b+1]*(1)+image[a][b-1]*(0)+image[a][b]*(0)+image[a][b+1]*(0)+image[a+1][b-1]*(-
1)+image[a+1][b]*(-2)+image[a+1][b+1]*(-1))/4);
```

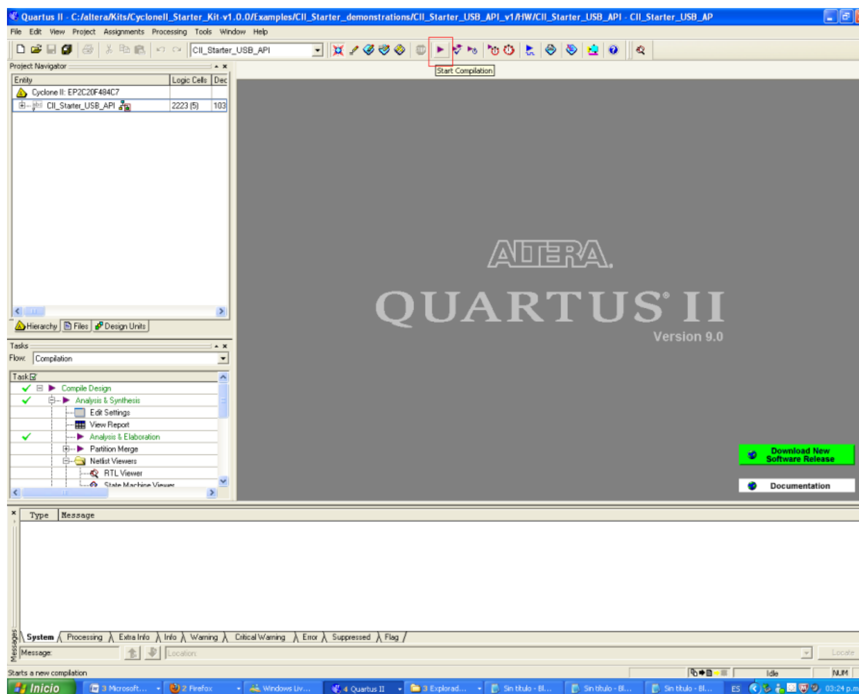
```
 }
}
  for(a = 2; a<= 479; a++){
     for(b = 2; b<= 639; b++){
   total1[a][b]=floor((mascarax[a][b]+mascaray[a][b])/2);
 }
}
for(b=1;b<=640;b++){
total1[1][b]=image[1][b];
total1[480][b]=image[480][b];
}

for(b=1;b<=480;b++){
total1[b][1]=image[b][1];
total1[b][640]=image[b][640];
}
  t_fin = clock();
  secs = (double)(t_fin - t_ini) / CLOCKS_PER_SEC;
  printf("%.16g milisegundos\n", secs * 1000.0);
pf = fopen("archivo_x.txt","w");
  for(a = 1; a<= 480; a++){
     for(b = 1; b<= 640; b++){
fprintf(pf, "%d\n", total1[a][b]);
}
}
fclose(pf);
  getchar();
  return 0;
}
```

Anexo C. Manual software Control panel para Altera® Cyclone II EP2C20F484C7N

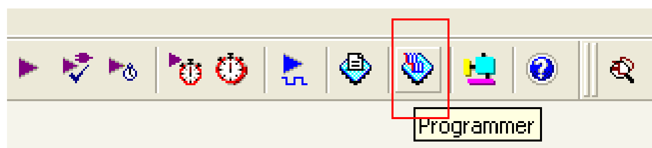
La tarjeta tiene como configuración determinada un demo en donde se puede observar el funcionamiento de los leds, display 7 segmentos y la salida VGA hacia un monitor. Ademas de esto la tarjeta contiene unos programas con los cuales se pueden realizar pruebas del funcionamiento de esta.En este tutorial se desea mostrar el funcionamiento del software Control Panel que viene con la tarjeta de desarrollo.

Después de instalar los drivers y el software que viene con la tarjeta de desarrollo debe ir a la carpeta donde quedaron instalados. Para este caso es C:\altera\Kits\CycloneII_Starter_Kit-v1.0.0. aca se encuentran los ejemplos que vienen en la tarjeta acceda a CII_Starter_demonstrations y luego a CII_Starter_USB_API_v1. Aca se encuentra la carpeta HW y SW, en SW se encuentra el software correspondiente al control panel y el pre procesamiento de imágenes que veremos posteriormente. Ingrese a HW abra el proyecto de Quartus llamado CII_Starter_USB_API.qpf. En esta parte le abrirá el proyecto y deberá compilarlo y grabarlo en la tarjeta. Debe conectar la tarjeta con el computador por medio del conector USB y encender la tarjeta, para poder grabar en la tarjeta.

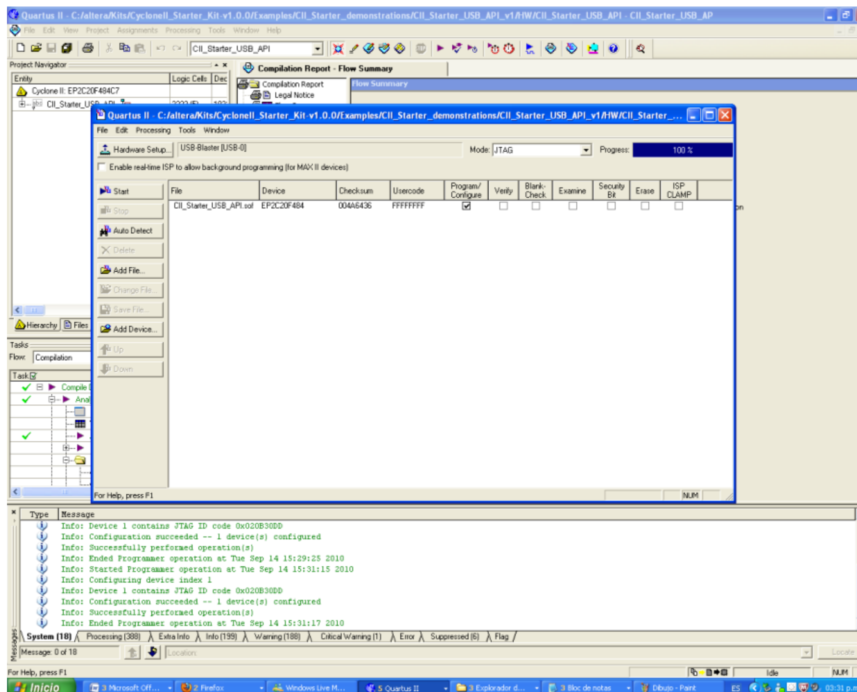
Ahora dele click al botón start compilation así:



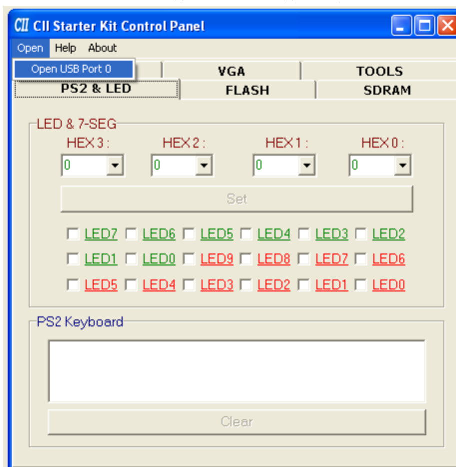
Y luego programmer



A continuación le abrirá la ventana del programmer en donde debe seleccionar en hardware setup USB-BLASTER y luego start



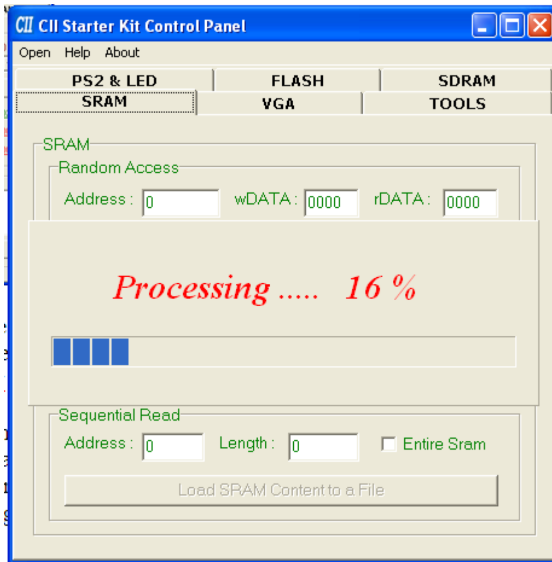
Ahora se encuentra configurada la tarjeta con el hardware adecuado para utilizar el control panel. Abra la carpeta SW y utilize el archivo CII_Starter_Kit_Control_Panel.exe
Dele click a la pestaña open y click en open port así:



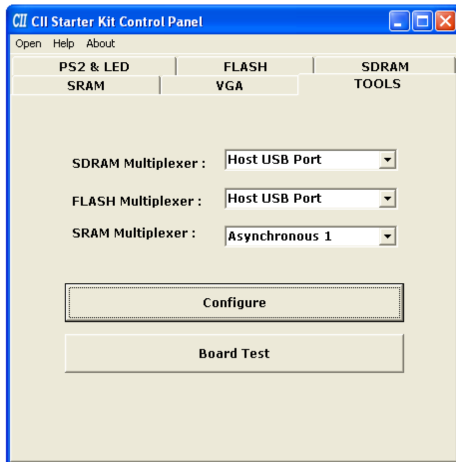
El control panel es un software que se utiliza para comunicarse con la tarjeta de desarrollo por medio de la conexión USB-BLASTER, aca se pueden configurar las memorias, los leds y el display de la VGA. En este tutorial se desea implementar el Control Panel para cambiar la imagen que muestra en la salida VGA. Para este caso se debe abrir un programa llamado, ImgConv que se encarga de codificar la imagen que se desea cargar en la tarjeta. Es importante que la imagen que se desea cargar sea una imagen de tipo .BMP de tamaño 640x480.

Abra el ImgConv y cargue la imagen que desea transferir a la tarjeta, una vez seleccionada click en Save Raw Data, el software genera unos archivos en la misma carpeta donde se encuentra la imagen.

El siguiente paso consiste en guardar la imagen codificada en la memoria sram por medio del control panel. Seleccione la pestaña sram, en la casilla de sequential write seleccione File Length, luego click en el botón write a file to SRAM y cargue el archivo que generó el image converter terminado en _Gray, en este caso genero un archivo llamado Raw_Data_Gray y abrir. La memoria empieza a almacenar la imagen por medio del USB-BLASTER.



Luego de que la imagen sea cargada en la SRAM debe ir a la pestaña TOOLS y en la opción SRAM multiplexer, seleccionar Asynchronous 1y luego seleccione configure de la siguiente manera:



Finalmente seleccione la pestaña VGA y deshabilite la opción Default Image. En este momento la imagen predeterminada cambia a la seleccionada por el usuario.

CII Starter Kit Control Panel
Open Help About
PS2 & LED
FLASH
SDRAM
SRAM
VGA
TOOLS
DE1 Board
Cyclone II FPGA Starter Board
Cursor X : 0
Cursor Y : 0
terasIC
www.terasic.com
Default Image
Cursor Enable

Anexo D. Interfaz para pruebas

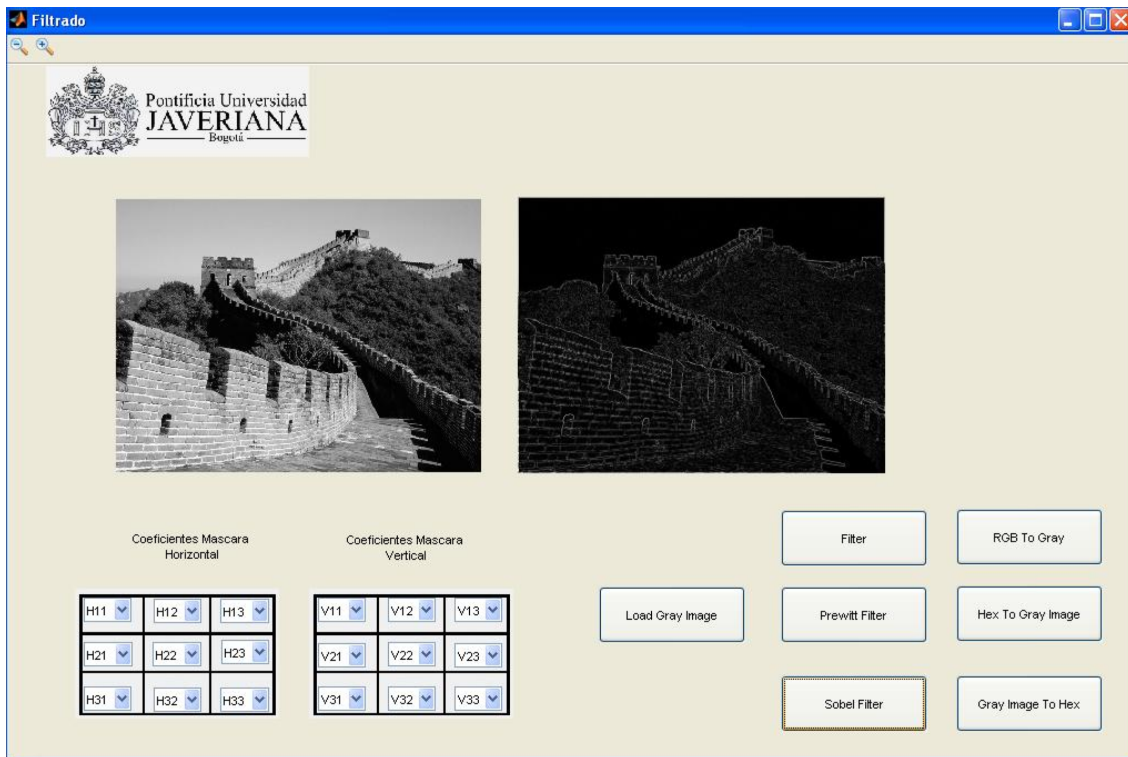


Figura Anexo A.Interfaz Grafica

La interfaz generada en MATLAB® realiza diversas utilidades para probar los resultados obtenidos en la FPGA. A continuación un listado y descripción de cada uno de los botones.

Load Gray Image: Carga una imagen de formato .bmp e ilustra la imagen en la interfaz.

Prewitt Filter: Realiza el algoritmo de Prewitt a la imagen cargada por el botón Load Gray Image.

Sobel Filter: Realiza el algoritmo de Sobel a la imagen cargada por el botón Load Gray Image.

Filter: Realiza un filtrado de la imagen cargada por Load Gray Image con los coeficientes que el usuario seleccionadas en el panel de las máscaras horizontal y vertical.

Memoria Externa SRAM		
Direccion SRAM	LowerByte	UpperByte
0	Pixel_0	Pixel_1
1	Pixel_2	Pixel_3
2	Pixel_4	Pixel_5
3	Pixel_6	Pixel_7
4	Pixel_8	Pixel_9
5	Pixel_10	Pixel_11
...	...	...
153596	307192	307193
153597	307194	307195
153598	307196	307197
153599	P_307198	P_307199

Almacenamiento Imagen En SRAM

Pixel_0	Pixel_1	...	Pixel_639
Pixel_640	Pixel_641	...	Pixel_1279
Pixel_1280	Pixel_1281	...	Pixel_1919
Pixel_1920	Pixel_1921	...	Pixel_2559
...	...	...	...
Pixel_305920	Pixel_305921	...	Pixel_306559
Pixel_306560	Pixel_306561	...	Pixel_307199

Imagen 640x480

Figura Anexo A

RGB To Gray: Convierte una imagen a color .bmp de 640x480 a escala de grises.

Hex To Gray Image: Convierte un archivo .Hex a .bmp.

Gray Image To Hex: convierte una imagen .bmp de escala de grises al formato .hex, que la FPGA reconoce como imagen a ser procesada.

Vale la pena aclarar la forma en que almacena los píxeles para que la memoria reconoce los pixeles en el formato .hex. La forma en que una imagen 640x480 administra los píxeles y como los almacena en el formato .hex, equivalente a las posiciones de almacenamiento en la memoria SRAM se ilustran en las figuras del presente anexo.