Universidad Michoacana

de San Nicolás de Hidalgo

Facultad de Ingeniería Eléctrica

Tesis:

"Diseño y Descripción de Herramientas Fundamentales para Implementar un Algoritmo de Superresolución en una FPGA"

Presenta:

Marcos Bladimir Romero Pérez

Para obtener el grado de:

INGENIERO EN ELECTRONICA

ASESOR:

M.C. Antonio Ulises Sáenz Trujillo

Morelia, Mich. Febrero del 2023

Agradecimientos

A Dios:
Por darme la vida, por estar en cada momento que lo necesite y por permitirme lograr una meta más en mi vida.

A mis padres:
José Romero Correa y María Dolores Pérez Guzmán
Gracias por permitirme ser parte de sus vidas, por el cariño que me han dado así también por el apoyo incondicional que me han brindado en cada una de las decisiones de mi vida, por las motivaciones, consejos y educación que me han convertido en la persona que soy actualmente.

A mis hermanos:
Betsaida Marisol, Mayra Adilene, Diego Edgar y Angel Aldaír, que siempre han estado conmigo apoyándome y a motivándome en cada uno de mis objetivos.

A mi asesor de tesis:
M.C. Antonio Ulises Sáenz Trujillo por ser partícipe de mi profesión, por ser uno de los mejores profesores que tuve a lo largo de este trayecto, por facilitarme conocimientos y aconsejarme respecto a mi investigación.

A mis amigos y amigas
Que me han apoyado y motivado durante el transcurso de los diferentes caminos que he tomado durante mi vida.

A mis profesores
Que me han transferido su conocimiento y sí también por generarme dudas, que me permitieron, conocer, explorar y conquistar planos del universo de la ciencia.

Dedicatoria

La presente tesis quiero dedicársela principalmente a mis padres ya que gracias a ellos resido en el cosmos en una conformación de los planos: físico, mental y espiritual, además que también es la consecuencia de todo el trabajo físico, que han realizado para que yo pudiera adquirir los conocimientos y recursos, para ultimar esta tesis. También quiero dedicársela a mis hermanos ya que junto a mis padres son la principal motivación para alcanzar propósitos tan espaciales como este. Además, se la dedico a mi ser ya que es el éxito del trabajo mental como físico y es una muestra de lo que soy capaz de lograr. Asimismo, dedicar todo ese trabajo físico y mental en la realización de esta tesis simboliza mi felicidad dado que especialmente todo ese trabajo es aplicado específicamente en las ramas de la ciencia que me interesan y a su vez radica en un mensaje hacia mi personalidad del futuro, el cual expresa que el éxito se adquiere en realización de todo eso que me gusta incluso si conlleva un esfuerzo del tipo que sea.

Resumen

En esta tesis se presenta una revisión del estado del arte respecto a técnicas y algoritmos para resolver el problema de la superresolución, y la descripción de un SOC (System On Chip) el cual es una herramienta fundamental para implementar un algoritmo especifico encontrado en dicha revisión.

La revisión del estado del arte está dividida en dos partes. La primera se enfoca en las propuestas de algoritmos por investigadores en el extranjero, mientras que la segunda parte se centra en las investigaciones a nivel nacional.

La descripción del SOC presentado es una descripción de hardware de un algoritmo que está basado en redes neuronales convolucionales. Dicho SOC esta descripto en el lenguaje *Verilog* y se le ha nombrado como ***Cap_Conv1*** **e** internamente pose un core que tiene como propósito realizar la operación de convolución, este procesador se le ha nombrado ***Ingrom*** y con una frecuencia de reloj de 50Mhz permite realizar dicha operación 373 veces más rápido qué la función conv2 de Matlab que fue ejecutada en un ordenador con un procesador i5 de tercera generación y 6GB de RAM.

Durante el desarrollo de esta tesis también se describió el hardware de una interfaz de imagen que permite extraer los canales de una imagen con el formato ***bitmap*** y esta interfaz es una herramienta fundamental que permite adaptarse a otros módulos digitales que necesiten manejar imágenes con este formato. Gracias a la facilidad de incorporación de este hardware se adaptó a ***Ingrom*** para describir un procesador de imágenes al que se le nombro ***PI*** y con una frecuencia de 50MHz realiza el procesamiento de una imagen 217 veces más rápido que Matlab, ejecutado en un ordenador con las características ya mencionadas. Así bien ***PI*** es una herramienta fundamental para resolver problemas del procesamiento digital de imágenes.

Palabras claves:

Hardware

Soc

Procesamiento

Imágenes

CNN

Abstract

This thesis presents a state of the art review of techniques and algorithms to solve the super-resolution problem, and the description of a SOC (System On Chip) which is a fundamental tool to implement a specific algorithm found in this review.

The review of the state of the art is divided into two parts. The first part focuses on algorithm proposals by researchers abroad, while the second part focuses on domestic research.

The description of the presented SOC is a hardware description of an algorithm that is based on convolutional neural networks. This SOC is described in the Verilog language and has been named Cap_Conv1 and internally has a core whose purpose is to perform the convolution operation, this processor has been named Ingrom and with a clock frequency of 50Mhz allows to perform this operation 373 times faster than the conv2 function of Matlab that was executed in a computer with a third generation i5 processor and 6GB of RAM.

During the development of this thesis we also described the hardware of an image interface that allows to extract the channels of an image with the bitmap format and this interface is a fundamental tool that allows to adapt to other digital modules that need to handle images with this format. Thanks to the ease of incorporation of this hardware, Ingrom was adapted to describe an image processor which was named PI and with a frequency of 50MHz performs the processing of an image 217 times faster than Matlab, executed on a computer with the aforementioned characteristics. Thus PI is a fundamental tool for solving digital image processing problems.

Contenido

Lista de Figuras

Lista de Tablas

Lista de Símbolos y Abreviaturas

SR	Superresolución
PDI	Procesamiento Digital de Imágenes
CNN	Red Neuronal Convolucional
HR	Alta Resolución
LR	Baja Resolución
IBP	*Iterative Back Projecting*
IKS	*Interative Kernel Steering*
SW	*Software*
HW	*Hardware*
PSNR	*Peak Signal to Noise Ratio*
PSF	*Point Spread Function*
HD	*High Definition*
RGB	*Rede Gran and Blue*
FPGA	*Field Programmable Gate Array*
DFT	*Discrete Fourier Transform*
CFT	*Continuous Fourier Transform*
SOC	*System On Chip*
FF	*Flip-Flop*
Cubesat	nano satélite
POCS	*Projection Onto ConvexSets*
DCNN	redes neuronales deconvolucionales
DNN	redes neuronales profundas
STV	método variacional
FHD	alta definición completa
UHD	ultra definición
FPS	Cuadros por segundo
IA	Inteligencia Artificial

CCD	Dispositivo de Carga Acoplada
CMOS	sensor de píxeles activos Complementario de Semiconductor de Óxido Metálico
SRCNN	Red Neuronal Convolucional para Superresolución
CLB	Bloques de Lógica Configurable
Tesela	Cada una de las piezas con que se forma un mosaico.
BMP	Bitmap
ISP	Procesador de imagen

Capítulo 1

Introducción

1.1 Introducción

Se usa el término de **superrresolución** (***SR)*** para describir el proceso de obtención de una imagen de alta resolución (HR, por sus siglas en inglés) o una secuencia de imágenes de HR a partir de una de imágenes de baja resolución (LR, por sus siglas en inglés) el cual se le conoce como proceso *mono-frame* o bien de un conjunto de imágenes LR, conocido como proceso múltiple-frame. Esto también se le denomina en la literatura ***mejora de la resolución*** [1].

A través de los años, el hombre se ha interesado en observar imágenes que le rodean, para posteriormente ilustrarlas, ya sea en pinturas rupestres en cavernas o bien en una pintura en un lienzo y expuesta en museos; sin embargo las ilustraciones en pinturas, no ha sido suficiente y se les ha adquirido en formato digital, mediante una cámara o algún otro instrumento y con el tiempo han mejorado las técnicas de adquisición, pero especialmente con la cámara digital no ha sido capaz de llegar a la calidad de percepción del ojo humano por lo que ha optado por desarrollar ciertas técnicas y algoritmos, como es la SR para intentar alcanzar esa calidad.

La SR ha ayudado parcialmente a alcanzar la calidad de la imagen, para que una vez que esta se ha adquirido, el ojo humano pueda interpretarla de mejor manera, sin embargo no ha sido su único objetivo, por lo que ha procesado y tratado para darle una visión a su tecnología desarrollada a través de los años y ha sido de gran impacto de tal grado que se le considerado una disciplina de la ciencia, conocida como visión artificial, donde también la calidad de la imagen es necesaria, para que dichas tecnologías puedan tener una mejor visión. Pero la necesidad de tener imágenes con SR y con mayor calidad no se limita en esta disciplina, sino que también se necesita en disciplinas como la medicina, astronomía, en aplicaciones tanto militares como civiles y de entretenimiento como, el cine, video juegos, etc.

1.2 Estado del Arte

El análisis que aquí se describe se realiza en dos grupos: la primera consta de investigaciones realizadas en diversos países y la segunda en investigaciones realizadas en diferentes estados del país.

1.2.1 INVESTIGACIONES EN EL AMBITO INTERNACIONAL:

“EVALUACIÓN DEL DESEMPEÑO DE MÉTODOS DE SR”

Camargo [2] en su tesis, menciona qué, aplica dos métodos para mejorar la resolución de imágenes médicas en 2D, los cuales son el método *Iterative Back Projecting* (IBP) y el método *Iterative Kernel Steering* (IKS) mediante *software* (SW). Su objetivo principal es mejorar la resolución de imágenes médicas hasta tres veces (3X).

Camargo, hace énfasis en las categorías y variables: método IBP, método IKS, mejora de la resolución de una imagen digital, ruido, submuestreo, desenfoque, interpolación de imágenes, Pico de Señal *Ratio–Ruido* o *Peak Signal to Noise Ratio* (PSNR), Punto de Función de Propagación o *Point Spread Function* (PSF), pruebas sintéticas, aplicaciones para el desarrollo nacional, aplicaciones en imágenes médicas, aplicaciones en imágenes satelitales.
Las técnicas y algoritmos que analiza y utiliza son: método *Back Projecting* (IBP), método Iterativo *Kernel Steering* (IKS), *Matlab*.

Como resultados Camargo menciona qué, el método de IBP tiene limitaciones para eliminar el ruido por lo que es necesario tener varias imágenes de la misma escena. En cambio, el método de IKS minimiza el ruido y es más eficiente en comparación del método IBP. Y concluye que logro mejorar la resolución de imágenes médicas hasta tres veces su resolución original (3x) y menciona que estos métodos pueden trabajar con cualquier tipo de imágenes, como puede ser imágenes satelitales.

“SUPERRESOLUCIÓN EN FPGAS”

Serrano [3] en su trabajo de fin de grado, menciona que las técnicas y los algoritmos para SR, se mantienen en constante desarrollo, pero no todas las técnicas y algoritmos son computacionalmente eficientes, por lo que selecciona trabajar con *Field Programmable Gate Array* (FPGAS) de esta manera intenta mejorar la eficiencia a nivel de descripción *Hardware* (HW) principalmente a que se ejecute de manera más rápida que la implementación de tales técnicas de SR en SW. Su objetivo es diseñar una aplicación de SR en un sistema embebido en este caso una FPGA. Estudio de alternativas y elección de algoritmos para su implementación en la FPGA.

Al escoger el algoritmo se implementará en lenguaje C para posteriormente implementarlo en la FPGA.

Serrano hace énfasis en las categorías y variables: SR, HR, LR, *High Definition* (HD), *Red Green and Blue* (RGB), FPGA, *Discrete Fourier Transform* (DFT), *Continuous Fourier Transform* (CFT), *System On Chip* (SOC), HW, SW, Flip-Flop (FF).

Las técnicas y algoritmos que analiza y utiliza son:

Dominio de la frecuencia:

La propiedad de desplazamiento de la transformada de Fourier.

La relación de los coeficientes con *aliasign* de la DFT.

Dominio Del Espacio:

Algoritmos no interactivos por interpolación:

Registro de la imagen de LR.

Interpolación.

Eliminación de desenfoque y de ruido.

Algoritmos de reconstrucción regularizada.

Enfoque determinista.

Enfoque Estocástico:

Maximum Likelihood.

Maximum a posteriori.

Algoritmos Pocs.

IBP.

Como resultados Serrano menciona que implementa el algoritmo en el lenguaje C para entender el algoritmo BSR y al implementar el algoritmo en una FPGA observo que es cuarenta veces más rápido (+40) con respecto a la implementación de SW, concluyendo que los algoritmos que mejor se adaptan a la tecnología de lógica programable, son los algoritmos no interactivos por interpolación.

"IMPLEMENTACIÓN Y EVALUACIÓN DE ALGORITMOS DE SUPERRESOLUCIÓN"

Jara [4] menciona en su tesis, que analizara algoritmos de SR, para mejorar la resolución de fotografías obtenidas por nano satélite (Cubesat), para ello usara el primer Cubesat chileno. Los algoritmos analizados los implementara en Matlab y los analizara visualmente para determinar cual tiene mejor resultado y analizara cual será más factible económicamente. Su objetivo general es evaluar el desempeño de algoritmos de SR para mejorar la resolución de imágenes obtenidas en dispositivos Cubesat.
Sus objetivos específicos son: implementar tres algoritmos de SR.
Evaluar cuál de los tres algoritmos es mejor para su Cubesat.
Analizar el impacto del uso del algoritmo de SR en el diseño de nuevos Cubesat.

Hace énfasis en las Categorías y variables: SR, algoritmos, imágenes.
Las técnicas y algoritmos analizadas y utilizadas son: IBP, *Projection Onto ConvexSets* (POCS), *ROBUST*, *Deep Learning*.

Como resultados Jara menciona que los algoritmos en cuestión de recursos computacionales el algoritmo POCS necesita menos memoria que el algoritmo IBP, sin embargo, el IBP tiene un tiempo ejecución más rápido.
Para el ROBUST tiene un consumo de memoria muy similar al de POCS, pero tiempo de ejecución significativamente menores.
También hace mención que los algoritmos *Learning* son mucho mejor ya que pueden aprender, sin embargo, hay que entrenar el algoritmo con escenas a las que va ser expuestas dicho algoritmo y definitivamente se va tener mejor calidad de imagen. Debido a que hay

que invertir tiempo en el entrenamiento opta mejor por los algoritmos mencionados anteriormente y ni si quiera implementa el algoritmo *Learning*. De los tres algoritmos que implementó observo que es el que tiene el peor resulta es el IBP en cuestión de la mejora de la resolución pero que utiliza menos recursos. En cambio, el algoritmo POCS mejora mucho la resolución, pero a un precio de ejecución muy lento, finalmente menciona que el algoritmo ROBUST es similar a las características del algoritmo POCS, pero con un poco más rápido y mejora la resolución y concluye que los algoritmos IBP, POCS, ROBUST, son algoritmos que se pueden implementar en los cubesat pero que cada uno tiene un factor de resolución diferente otros son más lentos que otros y que ocupan distintos tamaños de memoria. Finalmente prefiere el algoritmo ROBUST para implementarlo en su cubesat.

"OPTIMIZACIÓN DE CNN PARA SUPERRESOLUCIÓN DE IMÁGENES"

Chang y Kang [5] en su artículo, menciona que las CNN se utilizan en diversas aplicaciones de visión por computadora y se han realizados diversos estudios sobre las aceleraciones de CNN basados en FPGA para lograr un alto rendimiento y eficiencia energética. Mencionan que, hasta ese momento en la publicación del artículo, que pocas veces estas técnicas se han implementado para la mejora de la resolución de las imágenes, por lo que en este artículo proponen una red CNN rápida basada en FPGA para aumentar la resolución de imágenes de baja resolución, ya que las FPGAS cuentan una fuente de paralización por lo que puede realizar la ejecución de algoritmos rápidamente. Su objetivo es optimizar las CNN mediante FPGAS para tratar el problema de SR.

Hacen énfasis en las categorías y variables: SR, CNN, redes neuronales deconvolucionales (DCNN), FPGA. Las técnicas y algoritmos, analizadas y utilizadas son: redes neuronales convolucionales (CNN), DCNN, FPGA, *Xilinx*.

Como resultado los autores demostraron el rendimiento del acelerador propuesto en el FPGA *Xilinx Virtex-7 485T*. compararon el rendimiento del método TDC propuesto con el aselador DCNN. Implementaron la arquitectura propuesta. Los resultados experimentales mostraron que el método propuesto tenía un rendimiento hasta de 81 veces mayor que el método

convencional. Y concluyeron que lograron acelerar el aselador DCNN con el TDC. Para obtención de imágenes de HR a partir de imágenes LR, por lo que lograron que se acelera hasta 81 veces más rápido con el método DCNN convencional.

"SUPERRESOLUCIÓN DE IMÁGENES"

Marzoa [6] en su tesis de maestría, indica que en este trabajo se puede encontrar el estado de arte para algoritmos de SR de imágenes utilizando método *mono-frame* o *multi-frame*, además compara los métodos basados en redes neuronales profundas (DNN) con un método variacional (STV). De tal forma que estudia como impactan las diferentes técnicas en el mejoramiento de la resolución de imágenes de LR.

También menciona que para entrenar los modelos en CNN es necesario realizar el aprendizaje dándole como muestras de alta resolución, evalúa de como los métodos de sobre-muestreo y sub-muestreo afectan al aprendizaje de las CNN. Además, hace mención que trabajara con imágenes satelitales como imágenes a mejorar, finalmente proporciona el SW desarrollado en dicha investigación. Sus objetivos son busca y estudiar diferentes algoritmos de SR, en particular para usarlo en imágenes captadas por satélites.

Presentar el estado del arte para los algoritmos de SR tanto para algoritmos que utilizan una sola imagen o que utilizan varias imágenes. Estudiar algoritmos elegidos y aplicarlos a imágenes adquiridas por satélites.

Analizar la importancia de sub-muestreo con respecto a la elección con el núcleo de interpolación y como este proceso afecta al aprendizaje de las redes.

Implementar un programa de código abierto para mejorar la resolución.

La autora hace énfasis en categorías y variables: SR de imágenes, SR *mono-frame*, SR *multi-frame*, estimación de movimiento, redes neuronales, *DeepFeedforwardNetworks*, CNN, Redes Neuronales Recurrentes y Recursivas. Las Técnicas y algoritmos, analizadas y utilizadas son: Interpolación, SR basada en redes profundas, superresolución mono imagen, superresolución multi imagen, redes neuronales, *DeepFeedforwardNetworks*, CNN, Redes Neuronales Recurrentes y Recursivas, STV, interfaz desarrollada, Python, Opencv, Tkinter.

Como resultados la autora muestra en múltiples Figuras de como obtuvo el mejoramiento de la resolución de imágenes utilizando las dos técnicas de SR, es decir *multi-frame* y *mono-frame.* Y concluyó que se estudió el comportamiento de las DNN para el problema de SR de imágenes, presenta el estado de los algoritmos como los que utilizan una sola imagen de entrada o múltiples imágenes.

Compara los métodos basados en DNN con un método STV. Determina que cuándo se cuenta con las traslaciones de imágenes de alta precisión, los algoritmos convencionales presentan mejoras en el rendimiento que los algoritmos basados en CNN. En caso de lo contrario las redes proporciona mucho mejores resultados.

Para entrenar dichas redes es necesario generar muestras de imágenes de alta resolución.

Menciona que en caso de tener imágenes con ruido es necesario contar con redes entrenadas para este fin o bien aplicar filtros. Menciona que cuándo las redes neuronales ya son entrenadas dichos algoritmos son mucho más rápidos que los algoritmos convencionales STV, finalmente logra desarrollar la interfaz y la pone de manera libre.

"CNN PARA SUPERRESOLUCIÓN DE IMÁGENES"

Kim et al [7] indican en su artículo, que hasta donde conocen son los primeros en describir HW de una CNN en FPGA, en tiempo real la cual escala video de 2K de alta definición completa (FHD, por sus siglas en inglés) a video 4K de ultra definición (UHD, por sus siglas en inglés) a 60 cuadros por segundo (FPS, por sus siglas en inglés). En su sistema indican que el HW vasado en CNN, la entrada de LR, se procesa línea por línea, y el número de parámetros del filtro convolucional se reduce significativamente al incorporar convolucionales separadas en profundidad con conexión residual, su HW basado en CNN incorpora una cascada de convoluciones 1D que tiene grandes campos receptivos a lo largo de una línea horizontal mientras mantiene mínimos los campos receptivos verticales, para una implementación eficiente de HW, utilizaron un método de cuantificación simple y efectivo con poca degradación PSNR, además proponen un método eficiente para almacenar datos de los mapas, reduciendo el número de memoria de las líneas utilizadas en HW, la

CNN esta entrenada y probada en SW permitiendo ahorrar espacio. Sus objetivos son: Implementar una CNN en HW para aumentar la resolución de video de 2K a 4K con 60fps en tiempo real.

Los autores hacen énfasis en las categorías y variables: SR, FPGA, CNN, DNN. Las técnicas y algoritmos analizados y utilizados son: HW, FPGA, SW, redes neuronales profundas, CNN. Interpolación bicúbica.

Como resultados diseñaron una red SR compatible con HW basada en CNN, eficiente y novedosa que reconstruye video 4KUHD a partir de FHD a 60FPS. Concluyeron así la presentación de una red de SR compatible con HW basada en CNN que es eficiente y novedosa, hicieron que el HW pudiera reconstruir 4KUHD a partir de FHD a 60 FPS, su HW no utiliza ningún búfer de tramas y utiliza una pequeña cantidad de parámetros de filtro, redujeron la complejidad computacional y peso de memoria.

1.2.2 INVESTIGACIONES REALIZADAS EN EL ÁMBITO NACIONAL:

“SUPERRESOLUCIÓN DE IMÁGENES”

Souverville [8] menciona que, en su tesis, se verá el desarrollo de un algoritmo de SR basado en la teoría de lógica difusa implementado mediante SW Matlab. El algoritmo propuesto utiliza funciones de pertenencia Gaussianas para obtener imágenes de SR, también aborda las técnicas de SR de interpolación del vecino más cercano e interpolación bilineal. Su objetivo es: investigar, proponer y diseñar un algoritmo de SR para aumentar la resolución espacial en imágenes digitales, utilizando la técnica de lógica difusa mediante el uso de funciones de pertenencia Gaussianas. Evaluar el rendimiento y el tiempo de procesamiento del algoritmo propuesto mediante criterios de cuantificación usando el SW Matlab.

Souverville hace énfasis en las categorías y variables: SR, Interpolación, Imagen Digital, Modelo de color RGB, Lógica difusa. Las técnicas y algoritmos que analiza y utiliza son:

Método de interpolación, Método de interpolación del vecino más cercano, Método de interpolación bilineal, Matlab, Lógica difusa.

Como resultados el autor menciona que para cada uno de los algoritmos usados el tiempo promedio de procesamiento fue de 1.8 y 3.2 segundos. Y el autor concluye que es importante conocer la manera en cómo se conforma una imagen digital y como es su representación matemática para poder realizar el procesamiento de esta.

El algoritmo de interpolación de vecinos cercanos es un algoritmo que requiere poca carga computacional por lo que el tiempo de procesamiento es muy rápido. Su desventaja es que los resultados obtenidos presentan deformaciones en los bordes. En cambio, los algoritmos con lógica difusa tienen mejores resultados en cuestión de estos aspectos.

La propuesta en imágenes en escala de grises presenta conservación en los bordes y los detalles finos. En las imágenes de color no se corre con la misma suerte ya que hay una semejanza entre los canales de colores.

"SUPERRESOLUCIÓN DE IMÁGENES"

Rodríguez [9] presenta en su tesis, dos propuestas para resolver el problema de SR basada en única imagen, mejora la perdida de las frecuencias altas en las texturas, ya que los métodos tradicionales como interpolación lineal, bilineal o bicúbica tienen pérdidas de las frecuencias altas y en esta tesis se aborda la recuperación de dichas pérdidas mejorando estas técnicas.

Hace mención de que en base de una investigación puede mejorar la resolución de las imágenes a nivel de parches los cuales pueden representarse bien mediante técnicas de aprendizaje máquina, como máquinas de soporte vectorial, CNN, diccionarios y representación "sparse" y técnicas de agrupamiento.

Su objetivo principal es proponer un algoritmo que pueda resolver el problema de SR para una sola imagen de entrada, también tratara de recuperar frecuencias altas en zonas con textura, la cual dicha perdida se genera por los mismos algoritmos, mejorar el tiempo de ejecución del algoritmo.

Rodríguez hace énfasis en las categorías y variables: SR, *single image*, adquisición de una imagen, algoritmos basados en secuencias, algoritmos basados en una única imagen, LR, HR, CNN, algoritmo basado en diccionarios, entrenamiento de redes neuronales artificiales, entrenamiento de los diccionarios. Las técnicas y algoritmos, analizados y utilizados son: Redes neuronales artificiales, modelos basados en diccionarios, entrenamiento de redes neuronales, entrenamiento de diccionarios, estimación de parches de alta resolución, reconstrucción de imágenes de HR.

Los resultados que presenta Rodríguez en el capítulo cuatro, muestra varias imágenes como de entrada y de salida y las analiza visualmente, de esta manera se observa que efectivamente tiene mejoras en la resolución de la imagen de entrada. Y concluye que el primer modelo basado en redes neuronales artificiales es uno de los pocos trabajos realizados al momento de desarrollar la tesis. El segundo modelo propuesto está basado en conjunto de pares de diccionarios, donde dicho diccionario es estimado mediante un parche de baja resolución y otro con parches de HR y menciona que, a diferencia del primer método propuesto, ya hay bastantes trabajos. Los métodos propuestos en esta tesis hacen el uso directamente de agrupamiento para aprovechar la información para los parches. Menciona que esta técnica no es comúnmente vista en los trabajos del estado de arte. El agrupamiento efectivamente ayuda al problema de la SR y comenta que queda mucho por hacer.

El modelo basado en redes neuronales menciona es una idea muy simple como lo toma el autor ya que utiliza parches de alta resolución y baja de resolución e indica que entre más profundas sea dichas redes se tiene mejor resolución.

Con respecto al uso del modelo basados en pares diccionarios una de las importancias es usar de la norma Self-Similarity en relación con el factor de regulación al estimar los diccionarios ya que esto es poco común visto en los trabajos del estado del arte. Menciona que dichos trabajos proponen usar algoritmos de optimización y que las técnicas vistas en esta tesis son competitivas para los trabajos del estado del arte, finalmente menciona que para su caso no es necesario realizar un diccionario de LR y HR muy grande.

1.3 Objetivo

Diseñar y describir herramientas fundamentales para la implementar en una FPGA de un algoritmo de superresolución.

Objetivos particulares:

- Realizar la investigación de un algoritmo de superresolución y analizar si es factible de ser implementada en una FPGA.

- Diseñar y describir el HW de un procesador de convolución en lenguaje *Verilog*.

- Diseñar y describir el HW de una interfaz que permita leer una imagen con el formato BMP y que esta pueda ser procesada en una FPGA.

- Diseñar y describir el HW de SOC que permita realizar una capa de convolución para un algoritmo SR basado en redes neuronales convolucionales (SRCNN).

1.4 Justificación

A través de los años, la humanidad se ha interesado en plasmar imágenes que ilustren sus culturas, así como animales, plantas, paisajes, sus actividades, etc. para que trasciendan en los tiempos, cuya evolución ha ido desde simples garabatos para nuestra vista, hasta imágenes más complejas consideradas como obras de arte y que son más fácil de interpretar para el ojo humano.

Como sabemos nuestra evolución no se ha quedado en un mundo físico y hemos transcendido hacia un mundo digital y dicha transición no ha sido sencilla ya que de igual manera desde que se obtenía una imagen en una roca o un lienzo también en el mundo digital no se ha obtenido imágenes muy definidas inmediatamente por lo que poco a poco se ha ido mejorando el HW(sensores, transductores, lentes, etc.) para obtención de imágenes y que el ojo humano las pueda interpretar de mejor manera, sin embargo no ha sido suficiente y se

llegó a un punto donde el HW mencionado se ha quedado limitado principalmente por su alto costo y complejidad de fabricación para lograr obtener imágenes más nítidas para el ojo humano. Por ello es que surge la necesidad de lograr imágenes más nítidas al menor costo posible.

Para esto se han optado diversas técnicas principalmente con la ayuda del SW para mejorar la nitidez de una imagen como son los filtros digitales espaciales, los cuales hoy en día son muy conocidos y ampliamente usados en nuestros *Smartphone's*. Si bien con ellos podemos eliminar el ruido natural en una imagen digital, no mejora la resolución de una imagen y esta es una componente principal de la nitidez de una imagen por eso es que se necesita la SR de imágenes. Sin embargo, estas técnicas y algoritmos tienen un alto costo computacional y limita la obtención de una imagen rápidamente. Se conoce que con el paralelismo se puede realizar ejecución de instrucciones de forma más rápida por ello es que en esta tesis se pretende diseñar y describir una herramienta fundamental para inicializar la implementación de un algoritmo basado en CNN en una FPGA y de esta forma se pueda obtener una imagen de alta resolución en un menor tiempo.

1.5 Metodología

La presente tesis radica de una tesis del tipo teórico-practica, para cumplir los objetivos se siguió la siguiente metodología.

1. Se realizó una investigación a profundidad de diversas fuentes bibliográficas, con la finalidad de conocer las diferentes técnicas, algoritmos y contextos relacionados para resolver el problema de SR.
2. Mediante la investigación realizada, se seleccionó el algoritmo qué necesita las herramientas para poder ser descrito en HW.
3. Se diseño las herramientas fundamentales para el algoritmo seleccionado.
4. Se describió el HW de las herramientas fundamental, utilizando el *ISE* de Xilin, el lenguaje de descripción de HW *Verilog* y la plataforma *Matlab*.
5. Se comprobó el funcionamiento de la descripción de HW, con el simulador *Isim*, simulador *Modelsim* y la herramienta *RTL* propios del *ISE* de Xilinx.

6. Se puso a prueba la descripción de HW y se compró con los resultados obtenidos en diversas funciones tanto establecidas e implementadas en la plataforma *Matlab*.

1.6 Descripción de los capítulos

Capítulo 1 Introducción

En este capítulo se ha llevado a cabo la introducción de la SR de imágenes, la revisión del estado del arte de los diferentes métodos, técnicas y algoritmos para resolver el problema de SR. Además, se describe los objetivos, justificación, metodología y contenido de esta tesis.

Capítulo 2 Marco teórico

En este capítulo se describe el marco teórico, el cual se conoce los contextos fundamentales para conocer el mundo de las imágenes digitales, la SR de imágenes, la inteligencia artificial (IA) y el tipo de HW que se utilizara.

Capítulo 3 Arquitectura SRCNN

En este capítulo se expone la arquitectura de una CNN para SR de imágenes el cual requiere de la herramienta diseñada, la presentación se realiza tal cual como esta descrita en el artículo de Chao Dong, Chen Change Loy, Kaiming He y Xiaoou Tang, miembros de la IEEE.

Capítulo 4 Diseño y descripción de un procesador de convolución

En este capítulo se presenta el diseño y la descripción de HW de un procesador de convolución.

Capítulo 5 Diseño y descripción de HW de una interfaz de imagen

En este capítulo se presenta el diseño y la descripción de HW, que permite realizar la lectura de una imagen con el formato bitmap (BMP, por sus siglas en inglés) con la finalidad de que dicha lectura sea enviada a la descripción de HW del procesador de convolución.

Capitulo 6 Diseño y descripción de HW de un procesador de imágenes

En este capítulo se presenta el diseño y la descripción de HW de un procesador de imágenes.

Capitulo 7 Diseño descripción de una capa de convolución para el algoritmo SRCNN

En este capítulo se presenta el diseño y descripción del HW de SOC que permita realizar una capa de convolución para un algoritmo SRCNN.

Capítulo 8 Resultados

En este capítulo se presentan los resultados obtenidos del diseño y la descripción de HW de las herramientas fundamentales que requiere el algoritmo seleccionado.

Capítulo 9 Conclusiones y Trabajos Futuros

En este capítulo se presentan las conclusiones obtenidas en la realización de esta tesis, así como también se los trabajos futuros a desarrollar con la relación de esta investigación.

Capítulo 2

Marco Teórico

2.1 Imágenes Digitales.

2.1.1 Luz visible e invisible.

Aunque el campo del procesamiento de imágenes (PDI) se construye sobre una base de formulaciones matemáticas y probabilísticas, la intuición y el análisis humanos juegan un papel central en la elección de una técnica frente a otra, y esta elección a menudo es hecho sobre la base de juicios visuales subjetivos. Por lo tanto, es apropiado desarrollar una comprensión básica de la percepción visual humana por lo que estamos interesados en conocer las limitaciones físicas de la visión humana en términos de factores que también se utilizan en nuestro trabajo con imágenes digitales [10].

El ojo humano ve una parte del espectro de toda la luz que ilumina el universo. El rango de luz que podemos ver se denomina luz visible, es decir que hay longitudes de onda que no podemos ver, pero que existen, como, por ejemplo, la luz infrarroja (luz no visible por el ojo humano que está por debajo de la longitud de onda visible) y la ultravioleta (luz no visible por el ojo humano que está por encima de la longitud de onda visible) [11]. En la Figura 2.1, se puede apreciar una representación gráfica del espectro electromagnético, así como el rango de la longitud de onda visible por el ojo humano.

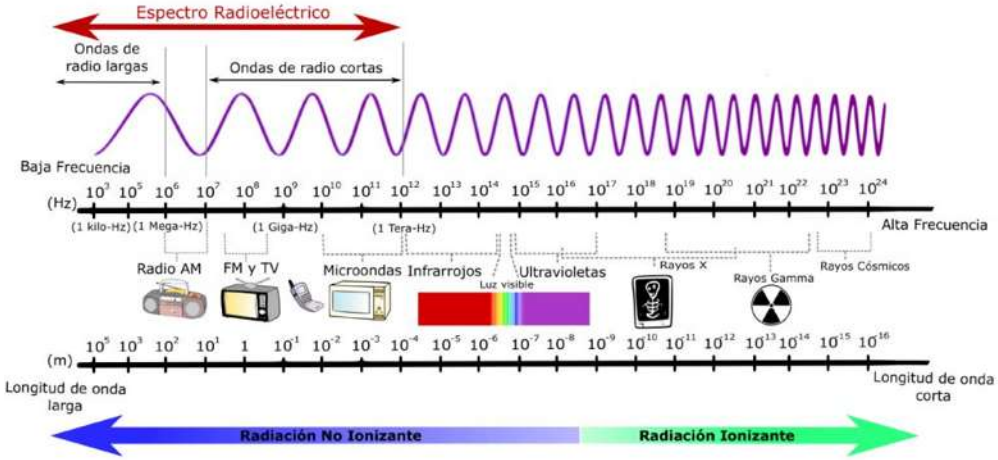


Figura 2.1. Espectro electromagnético [12].

El hecho es que la mayoría de sensores de cámaras digitales son sensibles a la luz visible, pero también son sensibles (en diferente medida) a la luz infrarroja y/o a la ultravioleta. Ahora bien, la luz infrarroja no nos es útil para construir una imagen digital, puesto que nos da información de una longitud de onda que no podemos ver y que, por lo tanto, no tiene una representación posible en un color [11].

2.1.2 Cámara

Los tipos de imágenes que nos interesan son generados por la combinación de una fuente de "iluminación" y la reflexión o absorción de energía de esa fuente por los elementos de la "escena" que es expuesta. Iluminación y escena entre comillas para enfatizar el hecho de que son considerablemente más generales que la situación familiar en la que una fuente de luz visible ilumina una escena tridimensional común y cotidiana. Por ejemplo, la iluminación puede provenir de una fuente de energía electromagnética como un radar, infrarrojos, o energía de rayos X. Pero, podría provenir de fuentes menos tradicionales, como el ultrasonido o incluso un patrón de iluminación generado por computadora. De manera similar, los elementos de la escena podrían ser objetos familiares, pero también pueden ser moléculas, formaciones rocosas enterradas o un cerebro humano. Incluso podríamos obtener imágenes de una fuente, como la adquisición de imágenes del sol. Dependiendo de la naturaleza de la fuente, la energía de iluminación se refleja o se transmite a través de los objetos. Un ejemplo en la primera categoría es la luz reflejada desde una superficie plana. Un ejemplo en la segunda categoría es cuándo los rayos X pasan a través de una superficie plana. cuerpo del paciente con el fin de generar una película de rayos X de diagnóstico. En algunas aplicaciones, la energía reflejada o transmitida se concentra en un foto convertidor (por ejemplo, una pantalla de fósforo), que convierte la energía en luz visible [13].

Mencionando lo anterior es necesaria una instrumentación la cual nos permita obtener una imagen tras captar la reflexión de luz, expuesta en una escena. El ejemplo de una cámara digital, así como sus partes se puede apreciar en la Figura 2.2.

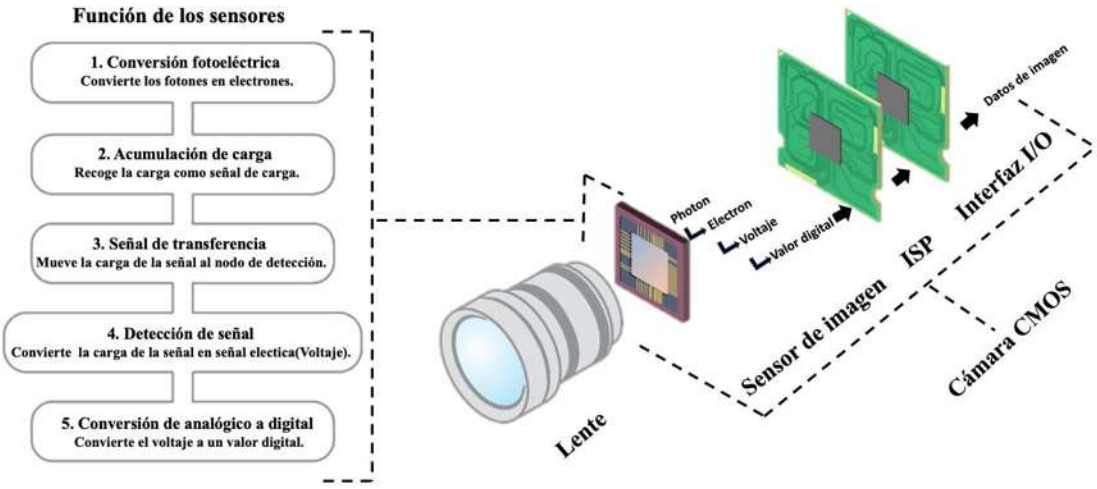


Figura 2.2, ejemplo de una cámara digital y sus partes [14].

Como se puede apreciar en la Figura 2.2, las cámaras digitales se pueden conectar directamente a computadoras por medio de una interfaz I/O, dicha interfaz pueden ser puertos USB, *FireWire*, HDMI, etc. Por lo cual nos permiten capturar imágenes en tiempo real [15].

2.1.3 Digitalización de imágenes.

Ahora bien, ya que sabemos con qué instrumentación podemos obtener una imagen tras captar la reflexión de luz, expuesta en una escena. Es importante conocer que proceso lleva la adquisición y como se compone una imagen digital.

El proceso de digitalización de una imagen consiste en convertir la imagen desde un formato continuo en el que existe en el mundo real a una representación numérica utilizable por la computadora [16].

- **MUESTREO**

  En primer lugar, es necesario discretizar el dominio espacial de la función de imagen $f(i,j)$. A la discretización (y limitación a un intervalo finito) de las coordenadas espaciales (i,j) se le denomina muestreo [17].

  El muestreo habitualmente se hace de forma que i y j tomen valores enteros, pero hay algunas diferencias de unos sistemas a otros [17]:

  - Hay sistemas que indizan los valores a ***i*** y ***j*** a partir de cero. Otros lo hacen a partir de uno.
  - Si bien la ***i***, que se destina a la coordenada horizontal, se hace comenzar siempre por la izquierda, la coordenada ***j*** que se destina a la vertical, en unos sistemas comienza por la parte inferior (la habitual en la representación cartesiana del primer cuadrante) y en otros por la superior (representación habitual para las matrices).

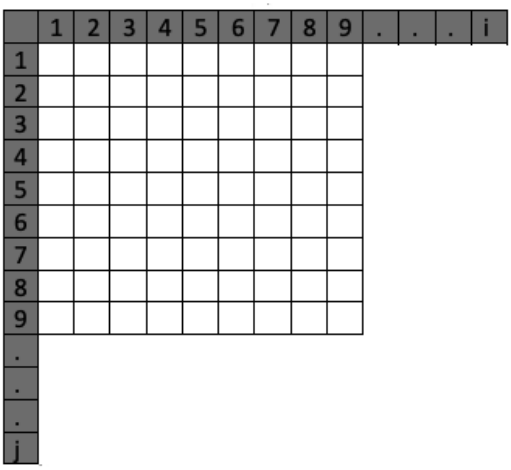


Figura 2.3, Muestreo de la imagen [17].

De esta forma, la función de imagen $f(i,j)$ se convierte en una matriz, la cual es una estructura fácilmente manipulable por un sistema informático [17].

$$f(i,j) = \begin{bmatrix} f(1,1) & f(1,2) & \cdots & f(1,M) \\ f(2,1) & f(2,2) & \cdots & f(2,M) \\ \vdots & \vdots & \ddots & \vdots \\ f(N,1) & f(N,2) & \cdots & f(N,M) \end{bmatrix}$$

Figura 2.4, representación matricial de la imagen [17].

Ahora podemos representar una imagen como un conjunto finito de $M \times N$ elementos, a cada uno de los cuales se le denomina píxel. En realidad, para cada píxel, se almacenará el valor medio o integral de la imagen en el rectángulo que lo constituye [17].

Un tema clave para la digitalización de la imagen es la selección de los valores de M y N pues determinarán un factor importante de la calidad de la imagen: la resolución espacial. También repercutirán en el espacio necesario para almacenar la imagen [17].

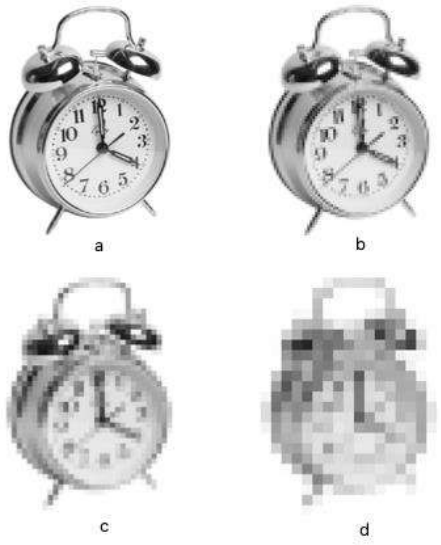


Figura 2.5, La misma imagen con diferentes muestreos [18].

En la Figura 2.5 se muestra la misma imagen con diferentes muestreos para: a: 300×300, b: 100×100, c: 50×50 y d: 25×25 [18].

- **Cuantización**

  El siguiente paso es la digitalización de la amplitud de la función de imagen, lo que se denomina cuantización. Para ello se ha de definir una forma de representación numérica adecuada a la información visual aportada por cada píxel. Dos son los tipos de imágenes que se pueden utilizar: de escala de grises o en color [18].

  - Imágenes a escala de grises

    Cuando la función de imagen mide los valores de intensidad luminosa, nos encontramos con imágenes en escala de grises y se puede representar con un valor numérico de esta magnitud por cada píxel [18].

    Aunque no es estrictamente necesario, sí es la costumbre utilizar números naturales para la cuantización de la intensidad lumínica o brillo. Los motivos fundamentales son dos: el número finito y no muy elevado de niveles producidos por los captadores físicos de imágenes y la considerable economía de almacenamiento de estos números frente a los reales. Además, y con el objetivo de adaptar lo mejor posible la escala al almacenamiento en la computadora, el número de niveles suele ser una potencia de 2, siendo la más frecuente de 2^8 = 256 niveles. También resulta interesante la escala de 2 niveles o blanco y negro [18].

Figura 2.6, Una misma imagen con 4 cuantizaciones distintas[10]

En la Figura 2.6, se muestra la misma imagen con cuatro distintas cuantizaciones $a: 256, b: 16, c: 4\ y\ d: 2$ para una escala de nivel de grises respectivamente al muestreo [19].

- **Imágenes en color**

  La representación del color se hace aprovechando las propiedades ópticas de la composición de la luz. Cualquier color visible por el ojo humano se puede expresar como composición de unos colores básicos [19].

  Cuando la combinación de los colores básicos se hace de forma que al ojo llega la suma de las componentes, nos encontramos con la composición aditiva y la base está formada por los colores rojo, verde y azul (RGB). Los tres juntos forman el blanco. Esta es la situación es la que se produce en las pantallas habitualmente utilizadas en las computadoras [19].

Figura 2.7, composición de colores RGB [19].

Si la combinación de los colores básicos se hace de forma que cada componente absorbe una parte de la luz blanca que llega hasta ellos, y solo lo no absorbido llega al ojo del observador, nos encontramos con la composición sustractiva, que es la que ocurre con las tintas de las impresoras en color. La base para este tipo de combinación está formada por los colores cian, magenta y amarillo (CMY) [19].

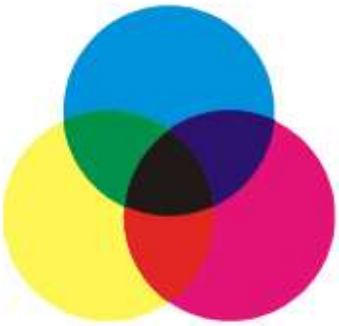


Figura 2.8, composición de colores CMY [19].

No son éstas las únicas bases posibles para definir el espacio del color. Aparte de éstas, y entre las más usadas, se encuentra el modelo de color HSL, que se basa en valores de tono, saturación y brillo. El tono es el color de base. La saturación es la pureza del color o la distancia que lo separa del gris y el brillo es la cantidad de blanco que contiene [19].

Todos los modelos de color existentes se relacionan entre sí y hay fórmulas que nos permiten pasar de uno a otro. Por ejemplo, entre RGB y CMY [19]:

$$C = MAX - R$$

$$M = MAX - G$$

$$Y = MAX - B$$

Otro ejemplo del modelo de color es el YCbCr, la relación matemática, que lo relaciona con el modelo de color RGB se expresa en la ecuación 2.1 [20].

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 + (65.481R + 128.553G + 24.966B) \\ 128 + (-37.797R - 74.203G + 120B) \\ 128 + (120R - 93.786G - 18.214B) \end{bmatrix} \qquad (2.1)$$

La ecuación 2.1, como ya se indicó se relaciona los dos modelos en el tiempo continuo. En cambio, la ecuación 2.2 representa la misma relación, pero para señales en el tiempo discreto.

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 + (65.481R + 128.553G + 24.966B)(\frac{1}{2^n}) \\ 128 + (-37.797R - 74.203G + 120B)(\frac{1}{2^n}) \\ 128 + (120R - 93.786G - 18.214B)(\frac{1}{2^n}) \end{bmatrix} \qquad (2.2)$$

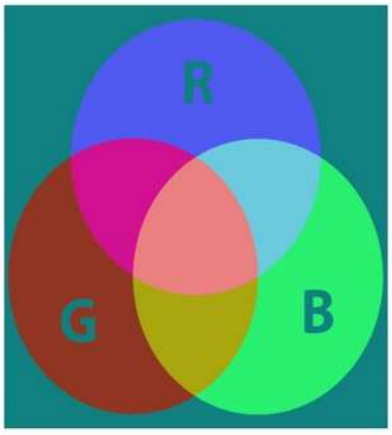


Figura 2.9, Relación del modelo de color RGB y YCbCr [21].

Una vez escogido un espacio de color, una imagen se representará indicando el color para cada uno de los pixeles. Para el almacenamiento, el modelo más utilizado es el RGB [19].

En las imágenes en color también se produce una cuantización, pues cada color se representa con una cierta precisión. De nuevo, se suelen utilizar números naturales para ello y, para adecuarlo al almacenamiento en la computadora, lo más frecuente es utilizar un byte para cada color de la base, lo que da un total de 3 bytes o 24 bits por píxel. El número de colores diferentes con esta cuantización es de $23 \times 8 = 16{,}777{,}215$. muchos más de los que el ojo humano puede distinguir, por lo que se denomina color real. También se utilizan cuantizaciones de 15 ó 16 bits por píxel [19].

- **Almacenamiento de imágenes**

  Internamente, cada programa almacena las imágenes de la forma que su diseñador ha decidido, aunque lo más frecuente es emplear la estructura matricial ofrecida por el lenguaje de programación utilizado.

  El tamaño necesario para almacenar una imagen depende directamente del muestreo y la cuantización. En general, una imagen de $M \times N$ píxeles, cada uno de los cuales se representa con b bits necesitará un espacio de M × N × b bits o, lo que es lo mismo, $(M \times N \times b) / 8\ bytes$ [19].

  Sin embargo, el tamaño final de un archivo conteniendo una imagen puede variar de un formato a otro, que se diferencian en varios factores [19].

  - Cabecera

    Cuando se almacena una imagen, normalmente se guarda con ella información adicional, como son el número de filas y columnas que contiene, cuantización utilizada, resolución a la que fue tomada, etc. Estos datos conforman la cabecera de la imagen, que suele encontrarse al principio del fichero [19].

    No hay grandes diferencias de un formato a otro en cuanto a los datos que forman la cabecera, pero sí en cuanto a la estructura, que es causa fundamental de incompatibilidades entre ellos.

- Paleta de colores

  El espacio necesario para almacenar un píxel no es mucho, pero si tenemos en cuenta la gran cantidad de píxeles que puede tener una imagen, puede resultar que el tamaño final resulte en ciertos casos excesivo. Por ejemplo, una imagen típica en color de $1{,}000 \times 1{,}000$ píxeles ocupan aproximadamente 3 Mbytes [19].

  Una estrategia para reducir el tamaño es la utilización de una paleta de color. Los colores disponibles para una imagen serán un subconjunto del espacio completo, normalmente escogidos entre los más usados en dicha imagen. De esta forma, y siempre que se incluya la paleta en el archivo de la imagen, para especificar el color de un píxel se indica el índice de la paleta donde se encuentra ese color. Nos encontramos así con imágenes definidas en un espacio de color amplio, pero del que solo usan un número reducido de colores [19].

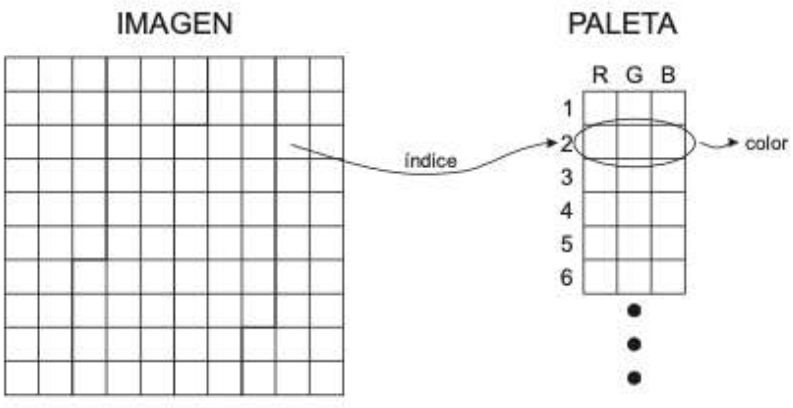


Figura 2.10, Determinación del color a través de la paleta [19].

Esto se hace en formatos para almacenamiento de imágenes, pero también se hace en el almacenamiento interno de algunos programas o sistemas gráficos, y a nivel de hardware como en algunos modos de visualización en la pantalla de la computadora [19].

Así, es frecuente encontrar imágenes con una paleta de 256 colores definida en un espacio de 16 millones de colores (color real) que ocupan poco más de la tercera parte que el original.

- Compresión

  Otra posibilidad para reducir el tamaño ocupado por una imagen es la utilización de técnicas de compresión en la propia especificación del formato de almacenamiento. En este aspecto hay que diferenciar dos tipos de compresión [19].

  Por un lado, se utilizan técnicas desarrolladas mediante teoría de información y aplicadas para comprimir cualquier tipo de datos y con las que la información, una vez restaurada es exactamente igual que la original. De este tipo, las compresiones más utilizadas son LZW y RLE [19].

  También se utilizan métodos de compresión diseñados específicamente para imágenes, que producen alteraciones en la imagen, a veces inapreciables, pero que a costa de ello pueden conseguir tasas de compresión espectaculares. Es el caso de la compresión JFIF usada en el formato JPEG [19].

2.2 Superresolución

2.2.1 Introducción

En la mayoría de las aplicaciones de imágenes digitales, generalmente se desean imágenes o videos de alta resolución para su posterior procesamiento y análisis de imágenes. El deseo de alta resolución se deriva de dos áreas de aplicación principales: mejora de las imágenes para interpretación humana; y ayudando a la representación para la percepción automática de la máquina. La resolución de la imagen describe los detalles contenidos en una imagen, cuanto mayor es la resolución, más detalles de la imagen. La resolución de una imagen digital se puede clasificar de muchas formas diferentes: resolución de píxeles, resolución espacial, resolución espectral, resolución temporal y resolución radiométrica. En este contexto, nos interesa principalmente la resolución espacial.

Resolución espacial: una imagen digital se compone de pequeños elementos de imagen llamados píxeles. La resolución espacial se refiere a la densidad de píxeles en una imagen y se mide en píxeles por unidad de área. La Figura 2.11, muestra un objetivo de prueba clásico para determinar la resolución espacial de un sistema de imágenes [22].

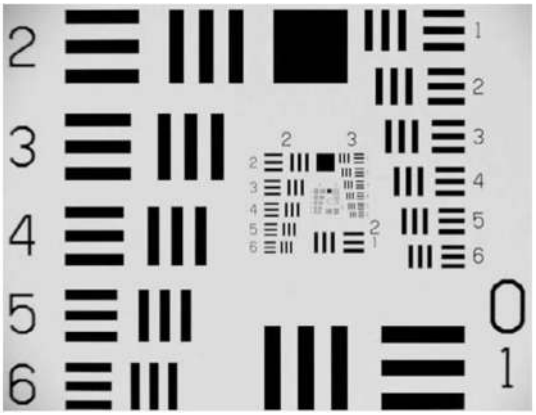


Figura 2.11, El objetivo de prueba de resolución de la USAF de 1951s [22].

En la Figura 2.11 se muestra un objetivo de prueba clásico utilizado para determinar la resolución espacial de sensores de imágenes y sistemas de imágenes [22].

La resolución espacial de la imagen está limitada en primer lugar por los sensores de imágenes o el dispositivo de adquisición de imágenes. Un sensor de imagen moderno suele ser un dispositivo de carga acoplada (CCD, por sus siglas en inglés) o un sensor de píxeles activos complementario de semiconductor de óxido metálico (CMOS, por sus siglas en inglés). Estos sensores suelen estar dispuestos en una matriz bidimensional para capturar señales de imagen bidimensionales. El tamaño del sensor o equivalentemente el número de elementos del sensor por unidad de área en primer lugar determina la resolución espacial de la imagen a capturar. A mayor densidad de los sensores, mayor resolución espacial posible del sistema de imágenes. Un sistema de imágenes con detectores inadecuados generará imágenes de baja resolución con efectos de bloque, debido a la baja frecuencia de muestreo espacial (*aliasing*). Para aumentar la resolución espacial de un sistema de imágenes, Una forma sencilla es aumentar la densidad del sensor reduciendo el tamaño del sensor mismo. Sin embargo, a medida que disminuye el tamaño del sensor, la cantidad de luz que incide en cada sensor también disminuye, provocando el llamado ruido de disparo. Además, el costo de hardware de un sensor aumenta con el aumento de la densidad del sensor o la correspondiente densidad de píxeles de la imagen. Por lo tanto, la limitación de hardware sobre el tamaño del sensor restringe la resolución espacial de una imagen que se puede capturar.

Mientras que los sensores de imagen limitan la resolución espacial de la imagen, los detalles de la imagen (bandas de alta frecuencia) también están limitados por la óptica, debido a los desenfoques de la lente (asociados con el PSF)), efectos de aberración de la lente, ¡di! raciones de apertura y borrosidad óptica debido al movimiento (trepidación). La construcción de chips de imágenes y componentes ópticos para capturar imágenes de muy alta resolución es prohibitivamente costosa y no es práctica en la mayoría de las aplicaciones reales, por ejemplo, cámaras de vigilancia y cámaras integradas en teléfonos celulares ampliamente utilizadas. Además del costo, la resolución de una cámara también está limitada en la velocidad de la cámara y el almacenamiento de hardware. En algunos otros escenarios, como las imágenes de satélite, es difícil utilizar sensores de alta resolución debido a limitaciones físicas. Otra forma de abordar este problema es aceptar las degradaciones de la imagen y utilizar el procesamiento de la señal para procesar posteriormente las imágenes capturadas,

para intercambiar el costo computacional con el costo del hardware. Estas técnicas se denominan específicamente reconstrucción de SR.

2.2.2 Definición

Se usa el término de ***SR*** para describir el proceso de obtención de una imagen de alta resolución (HR, por sus siglas en inglés) o una secuencia de imágenes de HR a partir de una de imágenes de baja resolución (LR, por sus siglas en inglés) el cual se le conoce como proceso *mono-frame* o bien de un conjunto de imágenes LR, conocido como proceso *múlti-frame*. Esto también se le denomina en la literatura ***mejora de la resolución*** [1].

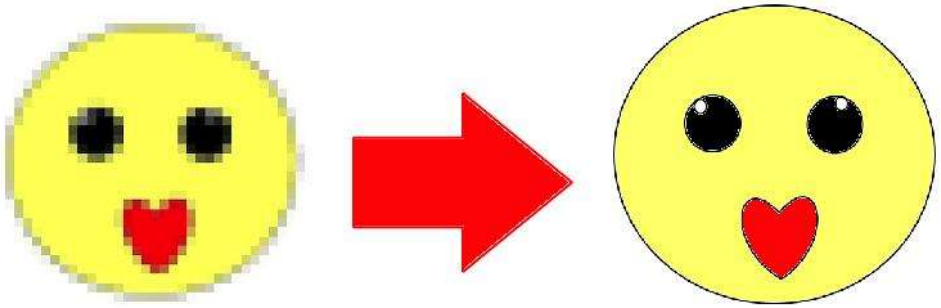


Figura 2.12, SR mono-frame.

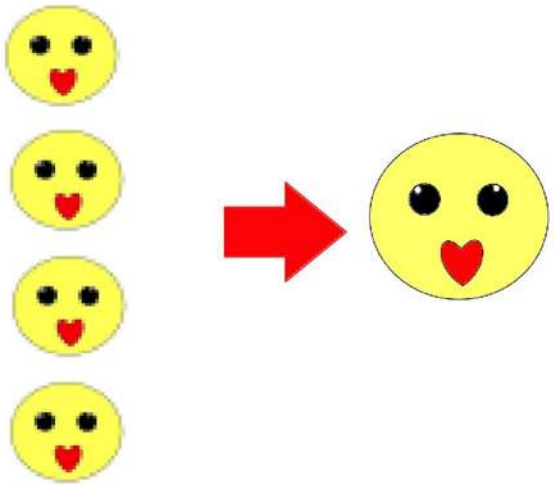


Figura 2.13. Multi-frame.

2.2.3 Aplicaciones:

El campo de la SR tiene un amplio campo de aplicación. Aunque el concepto de SR sigue siendo el mismo, las técnicas para lograr imágenes de recursos humanos pueden o no ser las mismas para todas y cada una de las aplicaciones. Esto se debe a que, en determinadas aplicaciones, como la **videovigilancia en tiempo real** o la **detección de objetivos**, el tiempo computacional es de gran importancia y, por lo tanto, requiere una técnica de SR con alta precisión y bajo costo computacional. Por otro lado, para ciertas aplicaciones, como **imágenes astronómicas** o **reconocimiento de texto**, el tiempo computacional no es una limitación y, por lo tanto, dichas aplicaciones pueden implementar técnicas de superresolución con alta precisión y un mayor costo computacional.

Para nombrar algunas aplicaciones de SR tanto en el dominio civil como en el militar [23]:

- Imágenes médicas.
- Sensores remotos.
- Detección y reconocimiento de objetivos.
- Imágenes de radar.
- Ciencia forense.
- Sistemas de vigilancia.

2.3 Inteligencia artificial

2.3.1 Introducción

Para abordar el concepto de IA, tal vez cabría plantearse primero la siguiente pregunta: **"¿qué es la inteligencia?"** Sin duda, se trata de una pregunta difícil cuya respuesta aún no ha sido resuelta totalmente, la cual sigue desconcertando tanto a los biólogos como a los psicólogos y filósofos de nuestra época.

Se podría comenzar por destacar algunas propiedades generales que presenta la inteligencia humana, como por ejemplo la habilidad de **enfrentar nuevas situaciones**, **la habilidad de resolver problemas, de responder preguntas, elaborar planes, etc**. Desde sus inicios, el hombre se representó el mundo real mediante símbolos, los cuales constituyen la base del lenguaje humano. En este sentido, se podría considerar a la IA como un dialecto simbólico constituido por cadenas de caracteres que representan conceptos del mundo real. De hecho, los procesos simbólicos son una característica esencial de la IA. A partir de lo expuesto es posible formular una definición más aproximada de nuestro objeto de estudio: la IA es una rama de las ciencias computacionales que se ocupa de los símbolos y métodos no algorítmicos para la resolución de problemas [24].

2.3.2 Ramas que componen la IA

Existen varios elementos que componen la ciencia de la IA, dentro de los cuales se pueden encontrar tres grandes ramas [24]:

- Lógica difusa
- Redes neurales artificiales
- Algoritmos genéticos

Cada una consta de características especiales, así como de una función específica. Sin embargo, la rama de nuestro interés son las redes neuronales artificiales y a continuación se expondrá dicha tecnología [24].

2.3.3 Redes neuronales artificiales.

- **Introducción**

  La tecnología neural trata de reproducir el proceso de solución de problemas del cerebro. Así como los humanos aplican el conocimiento ganado con la experiencia a nuevos problemas o situaciones, una red neural toma como ejemplos problemas resueltos para construir un sistema que toma decisiones y realiza clasificaciones. Los problemas adecuados para la solución neural son aquellos que no tienen solución computacional precisa o que requieren algoritmos muy extensos como en el caso del reconocimiento de imágenes [25].

  Las redes neurales se basan en generalizar información extraída de datos experimentales, tablas bibliográficas o bases de datos, los cuales se determinan por expertos humanos. Dichas redes neurales toman en cuenta las entradas (por ejemplo, corriente, voltaje, etc) y como salidas las señales del sistema (por ejemplo, velocidad, temperatura, torque, etc). La red neural utilizada es una red multicapa de diez neuronas en la capa de entrada, diez neuronas en la capa oculta y cinco neuronas en la capa de salida. Por lo tanto, se tienen 250 pesos ajustables mediante un control retroalimentado o de lazo cerrado. En la Figura 2.14 se presenta un diagrama de red neural. Los parámetros de inicialización se obtuvieron mediante un conjunto de datos experimentales y una base de datos [25].

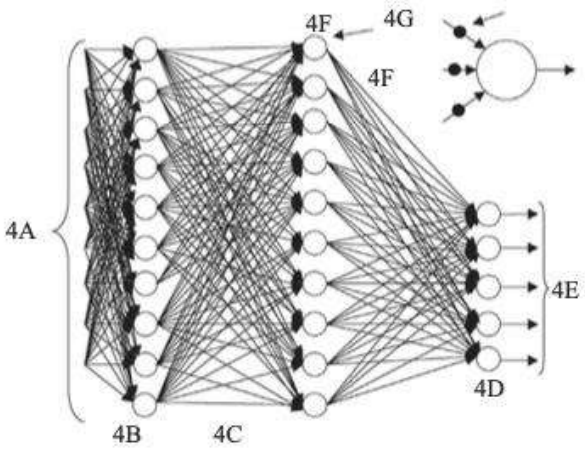


Figura 2.14, Esquema de la red neural multicapas 10-10-5 [25].

En la Figura 2.14 se muestra un esquema de red neuronal multicapa para el control inteligente con las entradas (4A): Representa cualquier variable. Capa de entradas (4B), capa oculta (4C) y capa de salidas (4D). Salidas (4E): Representa también cualquier variable de interés para el usuario. Los pesos entre cada neurona (4F) están representados por un punto negro (4G) [25].

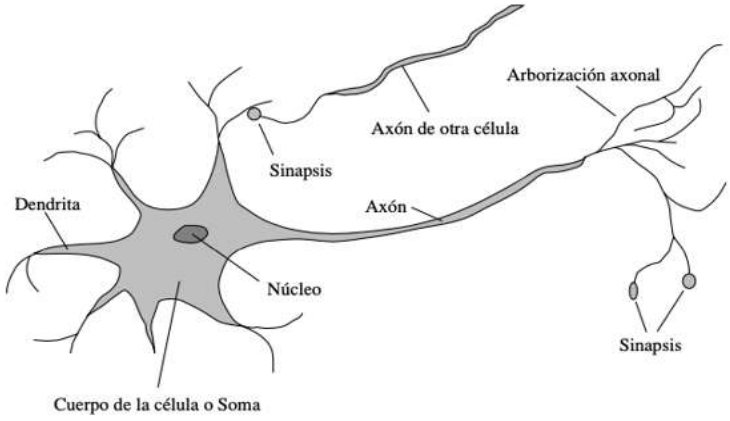


Figura 2.15, Partes de una célula nerviosa o neurona [26].

En la Figura 2.15, se muestra una neurona biológica que contiene un cuerpo celular, o soma, que tiene un núcleo celular. Un número de fibras llamadas dendritas se ramifican a partir del cuerpo de la célula junto con una única fibra larga llamada axón. El axón se alarga considerablemente, mucho más que lo que se representa en esta imagen. Normalmente miden un centímetro (100 veces más que el diámetro del cuerpo de la célula), pero pueden alcanzar hasta un metro de longitud. Una neurona se conecta con entre 10 y 100.000 neuronas formando una maraña llamada sinapsis. Las señales se propagan de neurona a neurona mediante una reacción electroquímica complicada. Las señales controlan la actividad del cerebro a corto plazo, y permiten establecer cambios en la posición y conectividad de las neuronas a largo plazo. Se piensa que estos mecanismos forman la base del aprendizaje del cerebro. La mayoría del procesamiento de información se lleva a cabo en la corteza del cerebro, la capa más externa de éste. La unidad de organización básica es una columna de tejido de aproximadamente 0,5 mm de diámetro, con lo cual se amplía la profundidad total de

la corteza cerebral, que en el caso de los humanos es de 4mm. Una columna contiene aproximadamente 20.000 neuronas [26].

El entrenamiento está basado en el algoritmo de "retro propagación del error" por el método del gradiente descendiente, en donde los pesos se actualizan mediante el uso de un conjunto ordenado de entradas y salidas deseadas y la comparación entre dicha salida y la salida real de la red neural. También se utiliza para el entrenamiento otra metodología alterna que es el "perceptrón". Es un clasificador de forma binaria: sólo existe la posibilidad de ser parte de un grupo A o B; funciona con sistemas lineales. Ambas tecnologías antes mencionadas se explican con más detalle a continuación [25].

- **Redes de retropropagación (*backpropagation*)**

  Principios para entrenar una red multicapa empleando el algoritmo de retro propagación:

  El algoritmo *Backpropagation* para redes multicapa es una generalización del algoritmo de mínimos cuadrados. Ambos algoritmos realizan su labor de actualización de pesos y ganancias con base en el error medio cuadrático. La red *Backpropagation* trabaja bajo aprendizaje supervisado y por tanto necesita un conjunto de instrucciones de entrenamiento que le describa cada salida y su valor de salida esperado. Si se considera la red de tres capas con dos entradas y una salida de la Figura 2.16, es posible apreciar que cada neurona está compuesta de dos unidades, donde la primera suma los productos de las entradas por sus respectivos pesos, y la segunda unidad contiene la función de activación (en la tabla 1, se puede apreciar las funciones de activación más usadas [27]). La señal y corresponde a la salida de la suma y 5 f (e) es la señal de salida del elemento no lineal de la función de activación, así como la salida de la neurona [25].

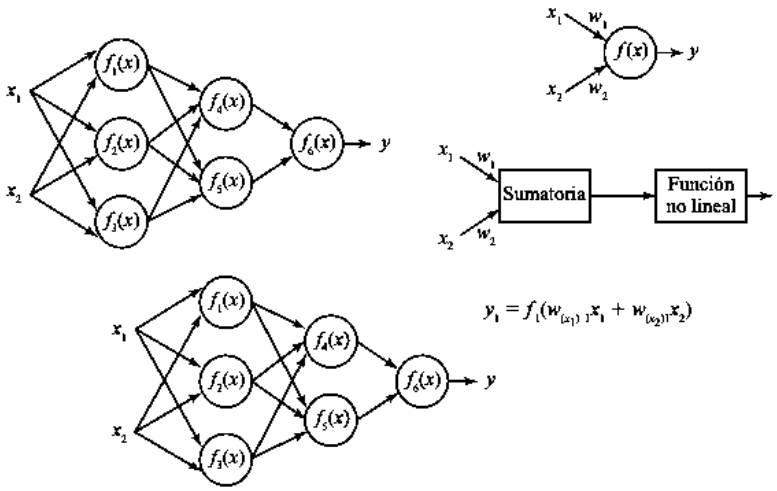
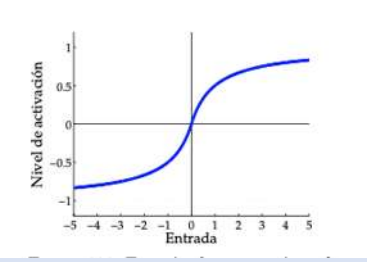
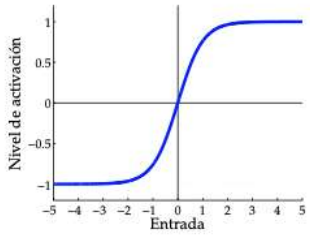
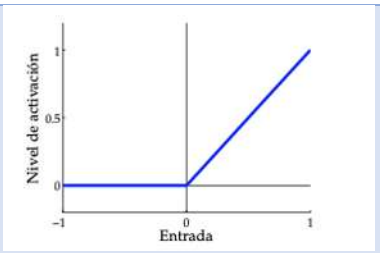


Figura 2.16, Red de tres capas [25].

Tabla 2.1, Funciones de activación más usadas [27].

Nombre	Función	Gráfica
función softsign	$fsoftsign(z) = \frac{Z}{1 + \lvert z \rvert}$	
Función tangente hiperbólica	$ftanh(z) = \tanh(z) = \frac{(e^{z} - e^{-z})}{e^{z} + e^{-z}} = \frac{1 - e^{-2x}}{1 - e^{-2x}}$	
Función Relu	$frelu(z) = z^{+} = max(0, z)$	

Para “enseñarle” a la red neural es necesario entrenar un conjunto de datos, el cual consiste en señales de entradas $x1$ y $x2$ asignadas con objetivos correspondientes (salidas deseadas) denominados z. El entrenamiento es un proceso iterativo. En cada iteración los pesos de los nodos se modifican usando nuevos datos del conjunto para el entrenamiento. Las modificaciones de los pesos se calculan empleando el algoritmo de retro propagación del error para el entrenamiento supervisado.

Cada paso del entrenamiento comienza al forzar ambas entradas de salida del conjunto de entrenamiento. Después es posible determinar los valores de salida de las señales de cada neurona en cada capa de la red. La Figura 2.17, muestra dos ejemplos de cómo se propaga la señal a través de la red, donde los pesos w_{mn} corresponden a la conexión de la salida de la neurona m con la entrada de la neurona n en la capa siguiente [25].

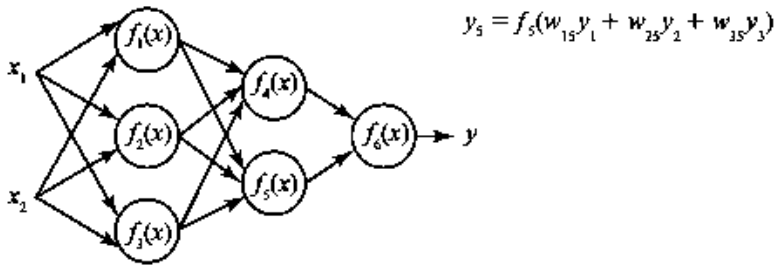


Figura 2.17, Propagación de las señales en las neuronas [25].

En el siguiente paso del algoritmo, la salida de la red se compara con el valor objetivo deseado. La diferencia se denomina error de la señal. Es imposible conocer el error en las neuronas de las capas internas directamente, debido a que los valores de salida de estas neuronas son desconocidos. El algoritmo de retro propagación propaga el error de regreso a todas las neuronas, cuya salida fue la entrada de la última neurona; esto se puede apreciar en la Figura 2.18.

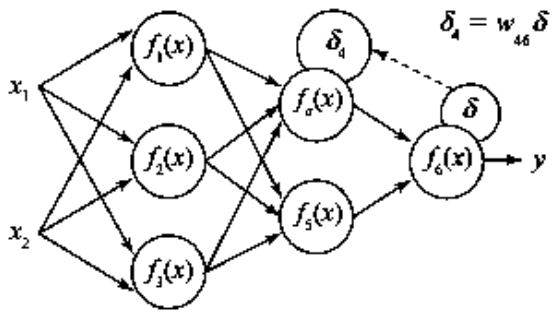


Figura 2.18, Retro propagación del error, capa de entrada, salida y ocultas [25].

Posteriormente el error se va propagando a las neuronas de capas anteriores considerando los pesos de las conexiones, según se muestra en la Figura 2.19.

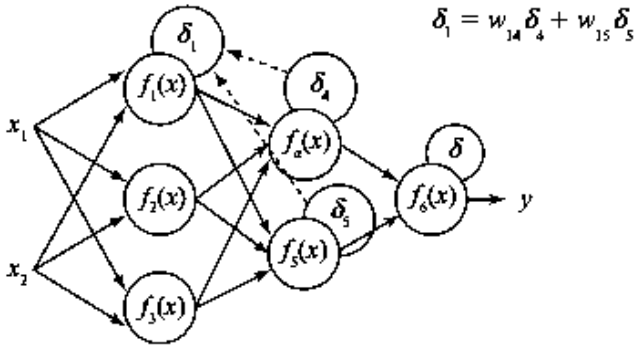


Figura 2.19, Retro propagación del error en capas intermedias [25].

Cuando se calcula el error para cada neurona, los pesos de entrada pueden modificarse según los ejemplos que se presentan en la Figura 2.20. Los coeficientes η afectan la velocidad de aprendizaje y pueden seleccionarse por distintos métodos. Uno de ellos implica que al inicio del proceso de entrenamiento se elige un valor grande, el cual va descendiendo gradualmente conforme avanza el proceso. Otro método comienza con parámetros pequeños que aumentan a medida que el proceso avanza y nuevamente disminuye en la etapa final. Comenzar el proceso con un parámetro pequeño permite el establecimiento de los signos de los pesos.

Finalmente, las acciones de control son sencillamente pasar los datos de salida a los dispositivos que se conecten o, en su caso, plantas virtuales previamente cargadas en el Sistema Didáctico de Control Inteligente Multipropósito [25].

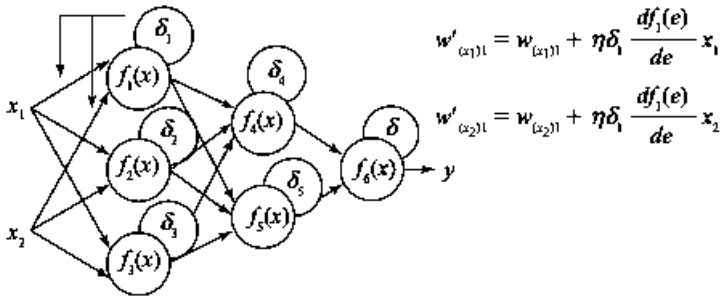


Figura 2.20, Actualización de los pesos [25].

2.4 Redes convolutivas o convolucionales

Si existe un tipo particular de red neuronal artificial que marca la diferencia en la práctica, ése es precisamente el correspondiente a las redes que se utilizan para procesar señales: las CNN. Su éxito en la resolución de problemas de visión artificial que, hasta hace poco, se consideraban casi intratables, sirvió para volver a poner de moda las redes neuronales en IA. Su explotación comercial dio lugar a lo que hoy entendemos por *Deep learning*. Tal como su propio nombre indica, las CNN se basan en el uso de convoluciones, una operación matemática familiar para todo el que haya trabajado en procesamiento digital de señales. Normalmente, se hace referencia a este tipo de redes mediante el acrónimo inglés CNN [*Convolutional Neural Network*] o, simplemente, con la composición acróstica *ConvNets* [28].

Las CNN son las redes neuronales artificiales que se utilizan habitualmente para resolver múltiples problemas prácticos que requieren procesar imágenes. Por ejemplo, cuándo la cámara frontal de un vehículo autónomo capta una señal de tráfico, debe identificar de qué señal concreta se trata (clasificación de imágenes). También puede interesarnos detectar qué tipos de objetos aparecen en la imagen correspondiente a una escena y localizarlos dentro de la imagen (detección de objetos). Incluso podemos utilizar redes convolutivas como parte de un sistema que genere una descripción textual del contenido de una imagen, con lo que podemos indexar imágenes para realizar búsquedas por contenido en bases de datos de imágenes o sintetizar una señal de voz a partir de la descripción textual para usuarios invidentes.

¿En qué se diferencian las CNN de las redes multicapa? Principalmente, en que tanto sus entradas como sus salidas pueden ser estructuradas. Las CNN nos permitirán aprovechar dicha estructura para diseñar arquitecturas especializadas que resuelvan de un modo más eficiente problemas que trabajen con tipos particulares de señales.

En lugar de recibir un vector de entradas correspondientes a diferentes variables (cuyas relaciones entre ellas desconocemos), recibiremos como entrada un vector (1D), matriz (2D) o tensor (>2D) en el que podemos explotar la relación física existente entre las diferentes entradas. En el caso de señales unidimensionales, puede tratarse de una señal de audio en el que entradas adyacentes corresponden a muestras consecutivas en el tiempo. En el caso de señales bidimensionales, las entradas pueden corresponder a los píxeles de una imagen captada con una cámara. También podemos tener señales bidimensionales de audio, como las provenientes de un *array* de micrófonos. En ocasiones, las señales de entrada puede que tengan más de dos dimensiones, como las imágenes en color (en la que tenemos imágenes individuales para distintos canales, como rojo, verde y azul), los datos volumétricos de imágenes médicas obtenidas mediante un escáner de tomografía computarizada o una resonancia magnética, o un simple vídeo (una secuencia de imágenes de dos dimensiones que incorpora también una dimensión temporal) [28].

Las redes convolutivas resultaron ser especialmente adecuadas para aprovechar la estructura bidimensional de los píxeles de una imagen. En las imágenes, suelen ser importantes las relaciones de adyacencia entre píxeles y las redes convolutivas explotan este hecho para obtener resultados que no se podrían conseguir con una red multicapa tradicional. Pero no sólo son útiles para imágenes, sino que también se han empleado con éxito en aplicaciones que trabajan sobre otros tipos de señales, como los sistemas de reconocimiento de voz [29].

2.4.1 La operación de convolución

Las CNN son, simplemente, redes multicapa en las que algunas capas realizan una operación de convolución en lugar de la tradicional multiplicación matricial de entradas por pesos. La convolución es una operación matemática que se realiza sobre dos funciones para producir una tercera que se suele interpretar como una versión modificada (filtrada) de una de las funciones originales. La convolución de las funciones f y g, que se suele denotar mediante un asterisco $*$ o una estrella $\star$, se define como la integral del producto de dos

funciones después de que una de ellas se refleje y se desplace [30], en la ecuación 2.1 se define la convolución en el dominio del tiempo continuo.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau = \int_{-\infty}^{\infty} f(t-\tau)g(\tau)d\tau \qquad (2.1)$$

En procesamiento digital de señales, cuándo utilizamos señales discretas, la integral anterior se convierte en una sumatoria, por lo que la ecuación 2.2, define la convolución en el tiempo discreto, tanto para señales causales y no causales.

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f(m)g[n-m] = \sum_{m=-\infty}^{\infty} f[n-m]g(m) \qquad (2.2)$$

Normalmente, uno de los operandos de la convolución es la señal que deseamos procesar, x$[n]$, y el otro corresponde al filtro, $h(n)$, con el que procesamos la señal. Cuando el filtro es finito y se define sólo sobre el dominio $\{0, 1, \dots, k-1\}$, la operación de convolución consiste en, para cada valor de la señal, realizar k multiplicaciones y $k-1$ sumas, la ecuación 2.3 expresa la operación de convolución en el tiempo discreto para señales causales.

$$(x * h)[n] = \sum_{k=0}^{k-1} h[k]x[n-k] \qquad (2.3)$$

La operación de convolución, que hasta ahora hemos definido sobre funciones de una variable, se puede extender fácilmente al caso multidimensional. En el caso de señales discretas definidas sobre dos variables a las que se aplica un filtro de tamaño $k1 \times k2$, la convolución se calcula utilizando la ecuación 2.4.

$$(x * h)[n1, n2] = \sum_{k1=0}^{k1-1} \sum_{k2=0}^{k2-1} h(k1, k2)x[n1-k1, n2-k2] \qquad (2.4)$$

La ecuación 2.4, expresa la convolución en el dominio espacial donde las variables $[n1, n2]$ corresponden a las coordenadas $[i, j]$, de una imagen.

2.4.2 Capas de *pooling*

Compartir parámetros a la vez que se usan campos receptivos locales permite controlar el número de parámetros de una red convolutiva. Pese a ello, el coste computacional de procesar imágenes en una red neuronal puede resultar bastante elevado. Por este motivo, las redes convolutivas suelen incluir un segundo componente clave: las capas de *pooling* o submuestreo [*subsampling*/*downsampling*] [31].

Básicamente, una capa de *pooling* lo que hace es reducir el tamaño de su entrada para que las capas posteriores puedan trabajar con datos de entrada reducidos. La reducción puede hacerse para reducir la dimensión espacial de los datos [spatial *pooling*] o para reducir su profundidad, el número de canales con los que hay que trabajar [*cross-channel pooling*]. La convolución 1×1 es, esencialmente, un mecanismo de pooling paramétrico que combina varios canales usando los pesos del kernel de convolución [31].

La versión tradicional de una capa de *pooling* se limita a combinar una serie de píxeles de entrada utilizando la función máxima, motivo por el que se denomina *max pooling*. Por ejemplo, si queremos reducir la dimensionalidad de una imagen, podemos hacer *max pooling* 2×2, que tesela la imagen en fragmentos de tamaño 2×2 y se queda únicamente con el máximo de cada fragmento para obtener una imagen de la mitad de ancho y de la mitad de alto, con una cuarta parte de los píxeles originales [31]:

$$\begin{bmatrix} 4 & 14 & 15 & 1 \\ 9 & 7 & 6 & 12 \\ 5 & 10 & 11 & 8 \\ 16 & 2 & 3 & 13 \end{bmatrix} \rightarrow \begin{bmatrix} 14 & 15 \\ 16 & 13 \end{bmatrix}$$

En una CNN, las capas de *pooling* se suelen utilizar intercaladas entre capas convolutivas. Dado que la salida de una capa de *pooling* pierde gran parte de los datos de su entrada, tal vez se esté preguntando qué sentido tiene agregar las salidas de receptores de características replicados. ¿No estamos desperdiciando el esfuerzo realizado por la capa convolutiva previa

a la capa de *pooling*? En realidad, una capa de *pooling* consigue dos objetivos simultáneamente, uno computacional y el otro práctico [31]:

- La capa de *pooling* reduce las dimensiones de las entradas que reciben las capas posteriores de una red neuronal. Esto nos permite, con una capacidad computacional limitada, ser capaces de calcular más características distintas en paralelo, al trabajar con entradas más pequeñas.
- El uso de capas de *pooling* proporciona una pequeña cantidad de invariancia frente a traslaciones en la señal de entrada. El uso de *pooling* hace que las señales con las que trabajan las capas posteriores a una capa de *pooling* sean aproximadamente invariantes frente a pequeñas traslaciones en la entrada.

2.4.3 Arquitectura de las redes convolutivas

Una CNN o *ConvNet*, está formada por una secuencia de capas. Todas las capas de la red transforman su entrada para generar una salida utilizando funciones diferenciables, lo que permite su entrenamiento con gradiente descendente y *backpropagation*. En general, en una CNN nos encontraremos tres tipos de capas: capas convolutivas, capas de *pooling* y capas completamente conectadas (como las utilizadas en redes neuronales tradicionales).

La arquitectura de CNN más común suele combinar capas convolutivas con capas de *pooling*. En ocasiones, entre la capa convolutiva y la capa de *pooling* se incluye una capa de unidades lineales rectificadas de tipo *ReLU*, que realiza una transformación no lineal de su entrada, elemento a elemento. Este patrón es tan común que, en ocasiones, las capas se contabilizan de forma diferente. Podemos contar las capas de una red convolucional de dos formas diferentes [32]:

Como una red formada por capas complejas que constan de tres etapas: convolución (operación lineal), detector *ReLU* (función no lineal) y *pooling*.

Como una red formada por capas simples consecutivas, algunas de las cuales (*ReLU* y *pooling*) ni siquiera tienen parámetros ajustables durante el entrenamiento de la red.

Esta estructura (*CONV → ReLU → POOL*) es habitual en redes convolucionales, en parte, porque nos permite establecer una analogía entre la topología de la red neuronal artificial y la estructura del córtex visual primario V1 [32]. En la Figura 2.21, se puede apreciar una arquitectura típica de red neuronal convolucionales para clasificación de imágenes.

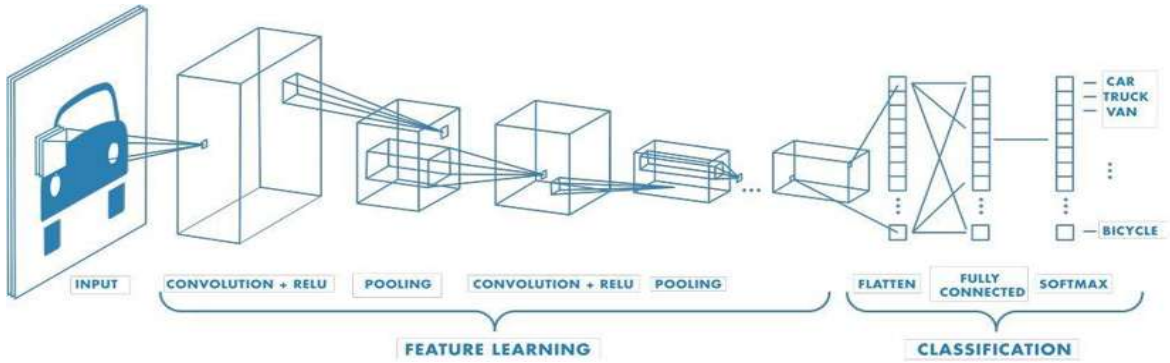


Figura 2.21, Arquitectura típica de red neuronal convolucional para clasificación de imágenes [33].

2.5 FPGA

2.5.1 ¿Qué es una FPGA?

FPGA por sus siglas en inglés ***Field programmable gate array*****, es un circuito integrado semiconductor que está estructurado mediante una matriz de bloques de lógica configurable (CLB),** en el que se puede cambiar la mayor parte de la funcionalidad eléctrica dentro del dispositivo; cambiado por el ingeniero de diseño, cambiado durante el proceso de montaje de la placa de circuito impreso, o incluso cambiado después de que el equipo ha sido enviado a los clientes en el "campo" [34].

2.5.2 Arquitectura FPGA

La estructura básica de una FPGA se compone de los siguientes elementos [35]:

Tabla de consulta (LUT): este elemento realiza operaciones lógicas.

Flip-Flop (FF): este elemento de registro almacena el resultado de la LUT.

Alambres(*wires*): estos elementos conectan elementos entre sí.

Pads de entrada/salida (I/O): estos puertos físicamente disponibles obtienen datos dentro y fuera de la FPGA.

La combinación de estos elementos da como resultado la arquitectura FPGA básica que se muestra en la Figura 2.22. Aunque esta estructura es suficiente para la implementación de cualquier algoritmo, la eficiencia de la implementación resultante es limitada en términos de rendimiento computacional, recursos requeridos y frecuencia de reloj alcanzable [35].

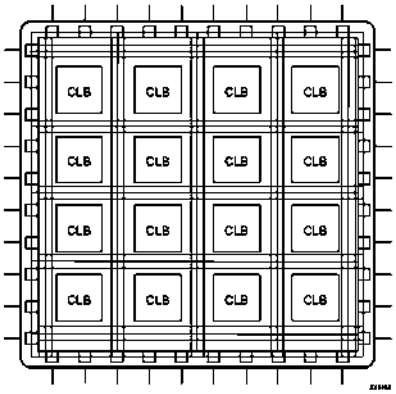


Figura 2.22, arquitectura básica de una FPGA [35].

2.5.3 Aplicaciones FPGA

Debido a su naturaleza programable, los FPGA son ideales para muchos mercados diferentes y aplicaciones como [36]:

- Aeroespacial y defensa.
- Automotriz.
- Datas Centers.
- Computación y almacenamiento de datos de alto rendimiento.
- Medicina.
- Seguridad.
- Procesamiento de video e imágenes.
- Comunicaciones cableadas.
- Comunicaciones inalámbricas.

Capítulo 3

Arquitectura SRCNN.

3.1 Introducción

La arquitectura que se presenta a continuación utiliza la técnica de mono frame, que como ya se presentó en el capítulo anterior esta técnica toma una sola imagen de baja resolución $x_{(i,j)}$, para convertirla en una imagen de alta resolución $F\{x\}$, donde F representa la función de transformación de la red neuronal convolucional. El objetivo del proceso de aprendizaje (optimización de parámetros) es lograr imágenes estimadas $F\{x\}$ tan parecidas como sea posible a las imágenes *ground truth* de alta resolución X_{HR}. Está compuesta por las tres operaciones siguientes [37]:

1. **Extracción y representación de parches.**
   Esta operación extrae parches(superpuestos) de la imagen de baja resolución $x(i,j)$, y representa cada parche como un vector de alta resolución. Estos vectores comprenden un conjunto de mapas de características, cuyo número es igual al número de operadores de representación (por ejemplo, filtros).
2. **Mapeado no lineal.**
   Esta operación mapea de forma no lineal cada vector de alta resolución en otro vector de alta resolución. Cada vector mapeado es conceptualmente la representación de un parche de alta resolución.
3. **Reconstrucción.**
   Esta operación agrega las representaciones de los parches de alta resolución anteriores para generar la imagen final de alta resolución. Se espera que esta imagen sea similar a la imagen *ground truth* $\boldsymbol{X}$.

En la Figura 3.1, se muestra una descripción general de la red y a continuación en las siguientes secciones se muestra la definición más detallada de cada operación.

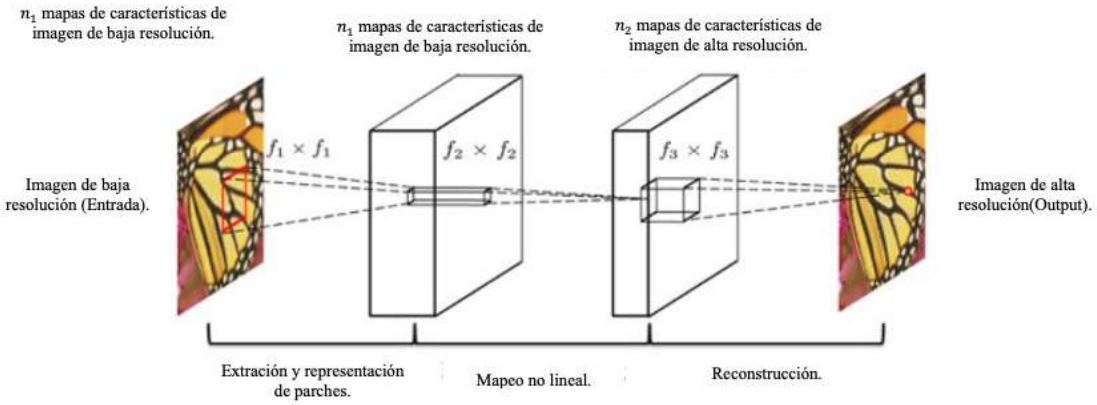


Figura 3.1. Arquitectura SRCNN [38].

En la Figura 3.1, se ilustran las siguientes etapas: Dada una imagen $x_{(i,j)}$ de baja resolución, la primera capa convolucional de la red extrae un conjunto de mapas de características. La segunda capa aplica una transformación no lineal a estos mapas de características a un nuevo espacio creando las representaciones de alta resolución de los parches. La última capa combina las predicciones ubicadas en regiones locales para producir la imagen final de alta resolución $F\{x\}$ [38].

3.2 Extracción y representación de parches

Una estrategia popular en la restauración de imágenes es extraer parches y representarlos mediante un conjunto de bases preparadas previamente como PCA, DCT, Haar, etc. Esto equivale a convolucionar la imagen mediante un conjunto de filtros, cada uno de los cuales es una base.

Formalmente, la primera capa de la red es expresada como una operación F_1 [37]:

$$F_1 = \max(0, W_1 * x + B_1) \qquad (3.1)$$

Donde W_1 y B_1 representan los filtros y los sesgos, respectivamente, y el * representa la operación de convolución. W_1 corresponde a n_1 filtros de tamaño $c \times f_1 \times f_1$, donde c es el número de canales de la imagen de entrada, y f_1 es el tamaño espacial del filtro. Intuitivamente W_1 aplica n_1 convoluciones a la imagen, y cada convolución tiene un kernel

de tamaño $c \times f_1 \times f_1$. La salida se compone de n_1 mapas de características. B_1 es un vector $n_{1\text{dimensional}}$, del que cada elemento está asociado a un filtro. Se aplica la función de activación ReLU ($\max(0, x)$) para rectificar los resultados de los filtros [37].

3.3 Mapeo no lineal

La primera capa extrae un vector de características $n_{\text{dimensional}}$ para cada parche. En la segunda operación, mapeamos cada uno de estos vectores $n_{\text{dimensionales}}$ en un vector $n_{\text{dimensional}}$.

Esto es equivalente a aplicar $n_2$filtros con soporte espacial trivial de $n_1 \times 1 \times 1$, que en la práctica consiste en una transformación no lineal del espacio de representación $n_{1\text{dimensional}}$ a otro de dimensión n_2. Esta interpretación sólo es válida para filtros de 1×1 , aunque la operación es fácil de generalizar a filtros más grandes como 3×3 o 5×5. En este caso, la transformación no lineal no se realiza sobre la imagen de entrada, sino sobre parches 3D del mapa de características generado por una capa anterior.

Formalmente, la segunda capa de la red es expresada como una operación F_2 [38]:

$$F_2\{x\} = \max(0, W_2 + F_1(x) + B_2) \qquad (3.2)$$

Donde W_2 contiene n_2 filtros de tamaño $c \times f_1 \times f_1$, y B_2 es n_2-dimensional. Cada uno de los vectores $n_{2\text{dimensionales}}$ de salida de la transformación no lineal es conceptualmente una representación de un parche de alta resolución que se usará para la reconstrucción [38].

Es posible agregar más capas convolucionales para aumentar la no linealidad. Pero esto puede aumentar la complejidad del modelo en $n_{i-1} \times f_i \times f_i \times n_i$ parámetros por cada nueva capa y, por lo tanto, exige más recursos de entrenamiento (tiempo, capacidad de cómputo, muestras de entrenamiento, etc.) [38].

3.4 Reconstrucción

En los métodos tradicionales, los parches superpuestos de alta resolución predichos a menudo se promedian para producir la imagen final completa. El promediado se puede

considerar como un filtro predefinido en un conjunto de mapas de características (donde cada posición es la forma vectorial "aplanada" de un parche de alta resolución). La ventaja de las DNN de extremo a extremo es que son capaces de aprender este filtro, adaptándolo a la naturaleza de los datos, en lugar de usar uno genérico.

Así pues, expresamos formalmente la última capa de la red como la salida de la misma [38]:

$$F\{x\} = \max(0, W_3 * F_2\{x\} + B_3) \qquad (3.2)$$

Donde W_3 corresponde a c filtros de tamaño $n_2 \times f_3 \times f_3$ y B_3 es un vector c-dimensional. El parámetro c corresponde al número de canales de la imagen de entrada; así, el modelo es capaz de reconstruir una imagen de salida con la estructura esperada [38].

3.5 Entrenamiento

Aprender la función de mapeado F consiste básicamente en estimar los parámetros de la red $\theta = \{W_1,\ W_2, W_3, B_1, B_2, B_3\}$ Esto se consigue a través de la minimización, mediante el algoritmo *backpropagation* (propagación hacia atrás), de una función de coste que calcula una medida de la diferencia entre las imágenes reconstruidas $F\{x; \theta\}$ (salida de la red) y las imágenes X de alta resolución o *ground truth* correspondientes [39].

3.5.1 Función de coste (MSE)

Dado un conjunto de imágenes de alta resolución $\{X_i\}$ y sus correspondientes imágenes de baja resolución $\{Y_i\}$, utilizamos el error cuadrático medio (MSE) como la función de coste [39]:

$$L(\theta) = \frac{1}{n} \| F\{x\} - X_{HR} \|^2 \qquad (3.3)$$

donde n es el número de muestras de entrenamiento [39].

3.5.2 Descenso de gradiente

El método de descenso de gradiente actualiza cada peso de la red en cada iteración de forma proporcional a la derivada parcial de la función de coste con respecto a dicho peso. Formalmente, esta función de actualización queda como sigue [39]:

$$\Delta_{l,i+1} = 0.9\Delta_i + \eta \frac{\delta L}{\delta W_i^l}, W_{i+1}^l = W_i^l + \Delta_{l,i+1} \qquad 3.4$$

Donde $l \in \{1,2,3\}$ e i son los indices de las capas e interacciones, η es el coeficiente del aprendizaje, y $\frac{\delta L}{\delta W_i^l}$ es la derivada parcial de L con respecto a W^l. La constante de 0.9 es el momentum que multiplica el cambio aplicado en la interacción anterior, que no es parte natural de descenso por el gradiente, pero que contribuye a la estabilidad del proceso de actualización de los parámetros [40].

Los pesos de filtro de cada capa se inicializan con una distribución Gaussiana de media 0 y desviación estándar 0.001, mientras que para la inicialización de las vías propone simplemente 0. El ratio de aprendizaje será 10^{-4} para las primeras dos capas y 10^{-5} para la tercera. Han demostrado empíricamente que aplicar un ratio de aprendizaje más pequeño en la última capa es importante para que la red converja [40].

3.5.3 Calidad de Reconstrucción

Finalmente se comprueba la calidad de la imagen de alta resolución, para ello se hace uso de las medidas de señal a ruido (SNR) las cuales son estimaciones de la calidad de una imagen reconstruida en comparación con una imagen original.

La idea básica es calcular un solo número que refleje la calidad de la imagen reconstruida [41].

Las imágenes reconstruidas con métricas más altas se juzgan mejor. De hecho, las medidas tradicionales de SNR no se equiparán con la percepción subjetiva humana. Varios grupos de investigación están trabajando en medidas de percepción, pero por ahora usaremos las medidas de señal a ruido porque son más fáciles de calcular. Solo recuerde que medidas más altas no siempre significan mejor calidad [41].

La métrica real que calcularemos es la medida de la señal de pico a la imagen reconstruida que se llama PSNR. Supongamos que se nos da una imagen fuente $f\ (i,j)$ que contiene M por N píxeles y una imagen reconstruida $F\ (i,j)$ donde F se reconstruye decodificando la versión codificada(bajando la resolución de la imagen intencionalmente) de $f\ (i,j)$. Las métricas de error se calculan en la señal de luminancia, por lo que los valores de píxel $f\ (i,j)$ oscilan entre negro (0)y blanco (255), para un esquema de color de 8 bits para el canal de luminosidad [41].

Primero, se calcula el error cuadrático medio (MSE) de la imagen reconstruida por lo que la ecuación 3.3, se reformula como se muestra en la ecuación 3.5.

$$MSE = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\|I(i,\mathrm{j}\,) - K(i-j)\|^2 \qquad 3.5$$

Posteriormente se calcula el PSNR en decibelio (dB), con la siguiente ecuación [41]:

$$PSNR = 10\log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20\log_{10}\frac{MAX_I}{\sqrt{\overline{MSE}}} \qquad 3.6$$

Donde MAX_I denota el máximo valor que puede tomar un píxel en la imagen, cuando éstos se representan usando n bits, $MAX_I = 2^n - 1$, es decir que, para la imagen de luminancia, los cuales sus pixeles oscilan entre negro (0) y blanco (255), para un esquema de color de 8 bits el $MAX_I = 2^8 - 1 = 256 - 1 = 255$.

Capítulo 4

Diseño y descripción de un procesador de convolución

4.1 Introducción

Como se indica en el capítulo 2.4, las CNN se basan principalmente en el uso de la operación de convolución, la cual esta descrita por la ecuación 2.3, para señales discretas unidimensionales y por la ecuación 2.4 para señales discretas bidimensionales. Además, la convolución es la principal operación que se utiliza en cada una de las capas de la arquitectura SRCNN presentada en el capítulo anterior; por ello es que en el presente capítulo se aborda la descripción de HW que permite realizar dicha operación.

4.2 Descripción de HW para realizar la operación de convolución espacial.

En el capítulo 2.5 se aborda el tipo de HW que se va utilizar el cual es una FPGA, que particularmente se trata de la FPGA *Spartan-6 XC6SLX9*, en donde se pretende implementar la descripción de la operación de la convolución.

La operación de nuestro interés es la convolución bidimensional o espacial, descrita por la ecuación 2.4 y antes de exponer la descripción de la convolución en nuestra FPGA es necesario mencionar y considerar ciertas características para realizar dicha operación.

- La primera característica consiste en que la imagen $\boldsymbol{x}(\boldsymbol{i},\boldsymbol{j})$ de tamaño $\boldsymbol{M}\times\boldsymbol{N}$ debe ser más grande que el filtro $\boldsymbol{h}(\boldsymbol{i},\boldsymbol{j})$, el cual también se le conoce como kernel o mascara de tamaño $\boldsymbol{m}\times\boldsymbol{n}$(en minúsculas para ser énfasis de que éste es menor a la imagen $\boldsymbol{x}(\boldsymbol{i},\boldsymbol{j})$), de aquí en adelante se referirá como kernel al filtro y se denotara de la siguiente forma $\boldsymbol{w}(\boldsymbol{i},\boldsymbol{j})$.
- La segunda característica consiste en considerar la inicialización de $(\boldsymbol{i},\boldsymbol{j})$ que como bien se menciona en el capítulo 2.1.3, hay sistemas donde inicializan en cero o en

uno. La descripción de HW para la operación de la convolución realizada considera, un sistema donde las variables $(\boldsymbol{i},\boldsymbol{j})$ inician en uno, esto sin importar el tamaño de la imagen, es decir que para las imágenes $\boldsymbol{x}(\boldsymbol{i},\boldsymbol{j})$ de tamaño $\mathbf{3}\times\mathbf{3}$ ilustradas en la Figura 4.1, se consideró la ilustración **a**.

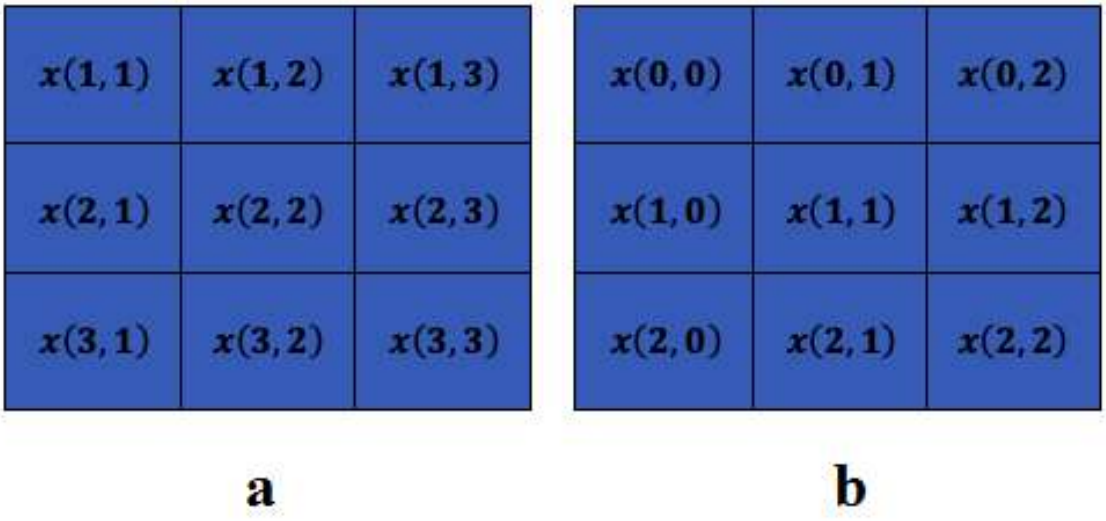


Figura 4.1, Inicialización de las variables $(\boldsymbol{i},\boldsymbol{j})$ para una imagen $x(\boldsymbol{i},\boldsymbol{j})$ de tamaño 3×3.

Una vez que se tomaron en cuenta las características anteriores, se diseñó el sistema que se muestra en la Figura 4.2.

Figura 4.2, Diseño del sistema de convolución.

Como se muestra en la Figura 4.2, el diseño del sistema consiste en recibir una imagen de tamaño $M\times N$ y un Kernel de tamaño $m\times n$, de tal forma que le bloque Conv2D realice la

operación de convolución espacial y como resultado obtener la imagen resultante, o también conocida como imagen filtrada o convolucionada, que se identificará como imagen filtrada.

Para afrontar la descripción de la convolución espacial, primeramente, se analizó la ecuación 2.4, y como se observa está compuesta de sumas ponderadas. Estas sumas ponderaras se pueden tratar directamente con bucles, los cuales son manejables en lenguajes de programación y lenguajes de descripción de HW como VHDL o Verilog, pero dichos bucles toman determinados ciclos de reloj y probablemente cuándo se trabaje con imágenes muy grandes el sistema puede hacerse lento.

Realizando el análisis anterior para afrontar la descripción, se percibió que el problema que causa afrontar la descripción de la convolución espacial con bucles, no resulta ser muy eficiente por lo que la siguiente forma de afrontarla, es realizar el proceso de convolución manualmente y hacer la descripción de hardware según las sumas, multiplicaciones y mapeo para cada valor de la matriz resultante, de esta forma nos aseguramos que una vez al describir el HW será más rápido que al realizar la descripción de HW con bucles. Sin embargo, se tardaría mucho tiempo en desarrollar la convolución espacial manualmente y probablemente sufrir errores.

Analizando las dos posibilidades anteriores de afrontar la descripción, se nota que no es posible obtener una forma eficiente tanto del costo computacional y el trabajo físico al describir el HW, por lo que se ha optado en realizar un hibrido de los dos análisis mencionados anteriormente, y que consiste en usar un lenguaje de programación haciendo el uso de bucles; de esta forma, se puedan generar literales de lenguaje Verilog tal cual como si realizara el proceso de la convolución manualmente para determinar la descripción de HW. Para ello se utilizó el lenguaje Matlab ya que posee una facilidad de trabajar con matrices. Las actividades realizadas para describir el diseño del sistema de la Figura 4.2 en Verilog se exponen en las siguientes secciones:

4.2.1 Descripción las entradas y las salidas

Se implementó en Matlab una función llamada *GenMatrices*, para facilitar la descripción de las entrada y salidas de sistema "*Conv2D"* ilustrado en la Figura 4.2. El código se muestra en la Figura 4.3.

```
function GenMatrices(m,n,V)
   for x=1:n
      for y=1:n
       fprintf('%c%d%d',V,x,y);
       if(x==m && y==n)
          fprintf(' ');
        else
          fprintf(', ');
       end
      end
   end
   fprintf('\n');
end
```

Figura 4.3, Código de la función GenMatrices.

Como se puede apreciar en la Figura 4.3, esta función recibe las variables **m**, **n** y **V** donde m y **n** son el tamaño de la matriz de la imagen, kernel o imagen resultante de nuestro sistema indicado en la Figura 4.2, la variable **V** recibe el valor de la variable con la que se desea trabajar, por ejemplo, para un kernel cuyo tamaño es de 3x3 y llamado *w* así como se ilustra en la Figura 4.4.

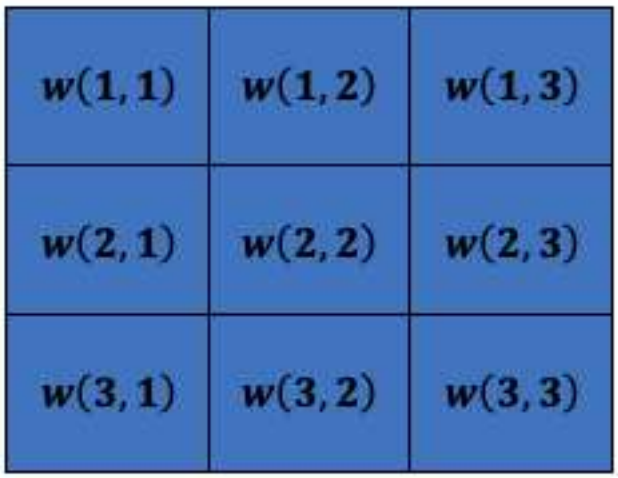


Figura 4.4. Kernel $w(i,j)$ de tamaño 3×3.

Se ejecutó la función *GenMatrices*, dicha ejecución se ilustra en la Figura 4.5.

```
>> GenMatrices(3,3, 'w')
w11, w12, w13, w21, w22, w23, w31, w32, w33
```

Figura 4.5. Ejecucion de la función GenMatrices.

Como se puede apreciar en la Figura 4.5, al ejecutar la función esta imprime la letra deseada con los valores de (i,j) hasta el tamaño de mxn y como vemos se parecen mucho a la estructura de la matriz kernel de la Figura 4.4, de esta manera ahorraremos tiempo al momento de describir hardware en *verilog* ya que directamente podemos definir si dichas variables son entradas o salidas, es decir podemos definir matrices de entrada o matrices de salida fácilmente.

4.2.2 Obtención de la descripción de la operación de convolución

Se implementó en Matlab la convolución espacial a nivel SW, para facilitar la descripción del HW y así conseguir que el sistema descrito no sea lento. La función implementada en Matlab es llamada Conv2D el código se muestra en la Figura 4.6.

```
function Conv2D(I,K,VF,VI,VK)
   b=size(I);
   c=size(K);
   r=ones(b(1),b(2));
   suma=0;
  for i0=2:1:b(1)-(c(1)-2)
   for j0=2:1:b(2)-(c(2)-2)
         suma=0;
         fprintf('%c%d%d<=(',VF,i0-1,j0-1);
     for i1=1:1:c(1)
      for j1=1:1:c(2)
         dif1=(i0+(c(1)-1)-i1);
         dif2=(j0+(c(2)-1)-j1);
       if (dif1>0 && dif2>0 && dif1<(b(1)+1) && dif2<(b(2)+1))
         fprintf('%c[%d][%d]*%c%d%d',VI,dif1,dif2,VK,i1,j1);
         suma=suma+(K(i1,j1)*I(dif1,dif2));
         r(i0,j0)=suma;
         if(i1==c(1) && j1==c(2))
         fprintf(');')
         else
           fprintf('+');
       end
      end
     end
    end
      fprintf('\n');
   end
  end
 fprintf('\n');
 disp(r);
 C=conv2(I,K,'valid');
 disp(C);
 X=conv2(I,K,'full');
 disp(X);

end
```

Figura 4.6. Código de la función Conv2D.

Como podemos observar en la Figura 4.6, la función Conv2D recibe como variables **I, K, VF**, **VI,** y **VK.** Donde la variable **I** es la posición donde se recibe una imagen de $M \times N$, la variable **K** dónde se recibe una matriz Kernel de $m \times n$, la variable **VF** es la letra que se desea que imprima para el resultado de la función de la convolución, es decir: $(x * h)[n1, n2]$ de la ecuación 2.4, la variable **VI** es la letra que se desea que imprima como variable de la imagen de la convolución y **VK** es la letra que desea que se imprime como valor del kernel. Se ejecuto el código para una imagen $x(i,j)$ de tamaño 5×5 y un kernel $w(i,j)$ de tamaño 3×3 y se obtuvieron las literales de lenguaje Verilog, que ayudaron a la descripción de HW para el modulo de convolución. Dichas literales se ilustran en la Figura 4.7.

```
F11<=(X33*W11+X32*W12+X31*W13+X23*W21+X22*W22+X21*W23+X13*W31+X12*W32+X11*W33);
F12<=(X34*W11+X33*W12+X32*W13+X24*W21+X23*W22+X22*W23+X14*W31+X13*W32+X12*W33);
F13<=(X35*W11+X34*W12+X33*W13+X25*W21+X24*W22+X23*W23+X15*W31+X14*W32+X13*W33);
F21<=(X43*W11+X42*W12+X41*W13+X33*W21+X32*W22+X31*W23+X23*W31+X22*W32+X21*W33);
F22<=(X44*W11+X43*W12+X42*W13+X34*W21+X33*W22+X32*W23+X24*W31+X23*W32+X22*W33);
F23<=(X45*W11+X44*W12+X43*W13+X35*W21+X34*W22+X33*W23+X25*W31+X24*W32+X23*W33);
F31<=(X53*W11+X52*W12+X51*W13+X43*W21+X42*W22+X41*W23+X33*W31+X32*W32+X31*W33);
F32<=(X54*W11+X53*W12+X52*W13+X44*W21+X43*W22+X42*W23+X34*W31+X33*W32+X32*W33);
F33<=(X55*W11+X54*W12+X53*W13+X45*W21+X44*W22+X43*W23+X35*W31+X34*W32+X33*W33);
```

Figura 4.7, Literales del lenguaje Verilog para describir la operación de convolución.

4.2.3 Obtención del módulo de convolución

Con la obtención de las literales de lenguaje *Verilog* ilustradas en la Figura 4.7, y con la ayuda de la función *GenMatrices*, se describió la convolución espacial en el lenguaje Verilog, utilizando la herramienta *ISE* 14.7 de *Xilinx* que soporta la FPGA *Spartan-6 XC6SLX9*. Una vez realizado la descripción HW en el lenguaje Verilog, con la ayuda de la herramienta *RTL* propia del *ISE* del *Xilinx* se obtuvo el sistema descrito. En la Figura 4.8, se puede observar el diagrama a bloque simplificado de la descripción de convolución.

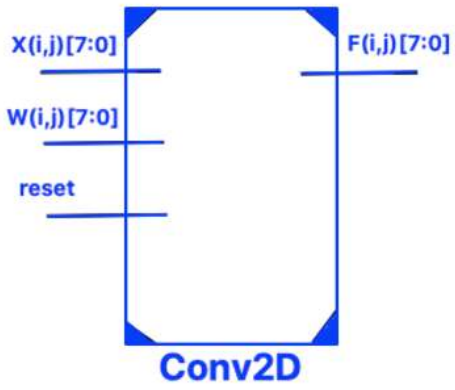


Figura 4.8, RTL simplificado de la descripción Conv2D.

El módulo Conv2D, permite realizar la operación de convolución con imágenes $\boldsymbol{X(i,j)}$ de un tamaño de $5x5$ y un kernel de $\boldsymbol{W(i,j)}$ de tamaño $3x3$. En la tabla 4.1, se ilustra más información del módulo Conv2D.

Tabla 4.1, Especificaciones del módulo Conv2D.

PINES.	Bus.	Función.	Características.
$\boldsymbol{X(i,j)}$	25×8 bits.	Son buses de entrada donde reciben el dato de cada pixel de la imagen.	El conjunto de estos buses recibe una imagen de tamaño de $5x5$ pixeles.
$\boldsymbol{W(i,j)}$	9×8 bits.	Son buses de entrada donde reciben el dato de cada pixel del Kernel.	El conjunto de estos buses recibe una Kernel de tamaño de $3x3$ pixeles y pueden ser datos con signo.
Reset	1 bit.	1 = no realiza la operación. 0 = realiza la operación.	La habilitación de la operación es sensible a flacos de bajada.
$\boldsymbol{F(i,j)}$	9×8 bits.	Se obtiene la imagen resultante o imagen filtrada.	Los datos obtenidos en estos buses pueden ser con signo o sin signo.

En la tabla 4.1, se describe las especificaciones del módulo Conv2D y como se puede ver los buses de las entradas que reciben los pixeles de la imagen y el kernel son de 8bits, debido a que las imágenes con las que se manejaran más adelante son imágenes con un esquema de color de 8 bits por canal.

4.3 Diseño de un procesador de convolución

En esta sección se presenta el diseño de un procesador de convolución dado que la operación de convolución descrita en HW, no es suficiente para utilizar imágenes reales, es decir la operación de convolución descrita anteriormente solo trabaja con imágenes $x(i,j)$ de tamaño 5×5 y un kernel de $w(i,j)$ de tamaño 3×3 y lo que se requiere es de que la operación pueda operar con imágenes de tamaño más grandes como por ejemplo de tamaño de 105×105.

4.3.1 Arquitectura básica de un procesador de propósito general

Antes de abordar el diseño del procesador de convolución, es necesario tener conocimientos previos de cómo se conforma un procesador de propósito general. La

arquitectura básica de un procesador está estructurada por cuatro secciones fundamentales las cuales son [42]:

- Una memoria principal.
- Una unidad lógica aritmética (ALU), capaz de realizar operaciones aritméticas.
- Una unidad de control.
- un equipo de entradas y salidas (E/S).

En la Figura 4.9, se ilustra la arquitectura de Von Neumann la cual es la arquitectura básica usada en la mayoría de los procesadores actualmente.

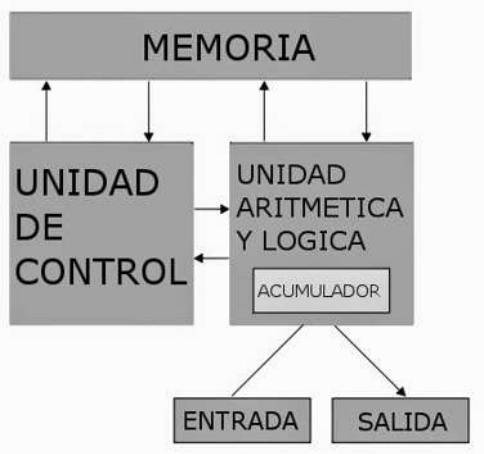


Figura 4.9, Arquitectura de Von Neumann [43].

4.3.2 Arquitectura de procesador de convolución

Teniendo el conocimiento de la arquitectura básica de un procesador de propósito general, se adquirió la arquitectura de un procesador de convolución el cual se ilustra en la Figura 4.10.

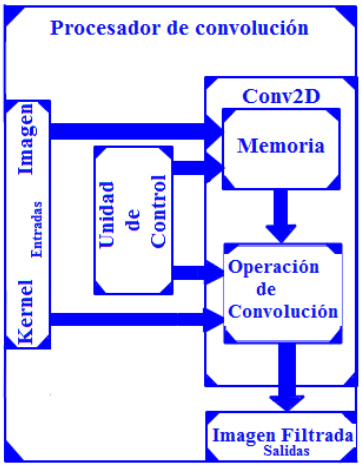


Figura 4.10, Arquitectura del procesador de convolución.

En la Figura 4.10, se ilustra la arquitectura del procesador de convolución; como se puede observar la arquitectura es muy similar a la arquitectura de la Figura 4.9, de igual forma ambas tienen una memoria principal, una unidad de control y bloques de entradas y salidas. La única diferencia es que la arquitectura de la Figura 4.9, tiene una unidad lógica aritmética en cambio la de la Figura 4.10, tiene una operación de convolución. A continuación, se expone el funcionamiento de cada unidad que contiene el procesador de convolución.

Iniciando con el módulo **Conv2D** el cual como se sabe este realiza la operación de convolución, pero note que la arquitectura ha cambiado un poco con respecto al de la Figura 4.2; ahora internamente tienen una memoria el cual tiene un tamaño de 5×5 y celdas de 8 bits, la cual es donde almacenara la imagen $x(i,j)$ de tamaño de 5×5, donde la unidad de control decide cuándo recibe la imagen introducida en el bloque de entradas(proceso de escritura de datos de entrada) y éste también decide cuándo los datos de la memoria son traslados a la unidad de la operación de convolución(proceso de lectura de la memoria), por ultimo una vez que se ejecuta la operación, se obtiene la imagen filtrada por el equipo de salidas.

Explicado lo anterior, inicialmente se puede ver que agregar una memoria interna al módulo de **Conv2D** no ayuda en nada, ya que se sigue operando con imágenes de tamaño $5x5$, sin embargo, sí ayuda bastante, ya que la imagen obtenida por el bloque de entradas será adquirida mediante una **extracción de parches** de una imagen más grande, aunque este

proceso ya se mencionó en el capítulo tres, en la siguiente sección se explica a detalle en que consiste.

4.3.3 Extracción de parches.

Como ya se sabe nuestro propósito es poder manejar imágenes con un tamaño más grande las cual puede ser una imágen de 105×105 pixeles, por lo que el proceso de extracción de parches consiste en tomar secciones de una imagen que se desea manejar. Por ejemplo, se tiene la imagen, como la ilustrada en la Figura 4.11, la cual se desea manejar en el procesador de convolución, así que a la Figura 4.11, se le extraerá secciones al tamaño de la memoria interna del módulo conv2D; es decir secciones de tamaño de 5×5, así como se ilustra en la Figura 4.12.

Figura 4.11, Imagen que se desea manejar.

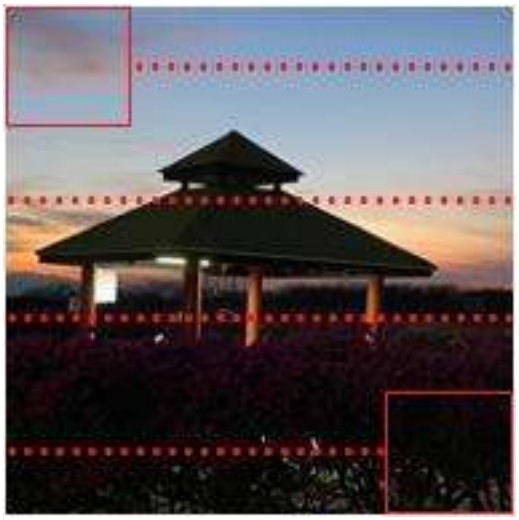


Figura 4.12, Representación gráfica de la extracción de parches.

En la Figura 4.12, se ilustra la representación de la extracción de parches, la cual como se observa, los cuadros de color rojo representan una extracción de la imagen de tamaño de la memoria, en este caso de un tamaño de 5×5 pixeles. La extracción inicia en la esquina superior izquierda y esta se va recorriendo hacia la derecha hasta llegar a la última extracción, es decir el segundo cuadrado posicionado en la esquina inferior derecha.

Esta ilustración solo es una representación gráfica de la extracción de parches, hay que tener en cuenta que las imágenes que manejara el procesador de convolución son imágenes a color, por lo que este proceso, **debe ser realizado por canal de color de la imagen.**

4.4 Descripción de HW del procesador de convolución

En esta sección se presenta la descripción de HW en lenguaje Verilog del procesador de convolución, así como cada uno de los módulos que lo componen.

4.4.1 Módulo Conv2D

De acuerdo a lo que ya se planteó, se conoce que al módulo de Conv2D, descrito en la sección 4.2.3, se le agrego una memoria interna, por lo que la descripción de éste se modificó y se obtuvo el módulo que se ilustra en la Figura 4.13.

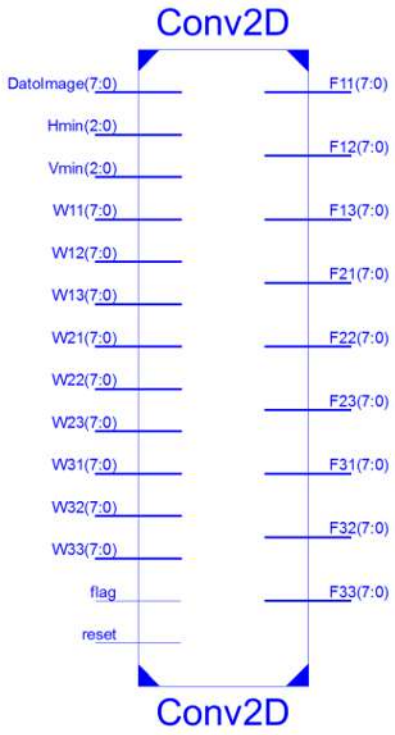


Figura 4.13, Módulo Conv2D modificado.

En la Figura 4.13, se ilustra el módulo de Conv2D, que si lo comparamos con el de la Figura 4.8, se puede observar que ya no aparecen las entradas de la variable $\boldsymbol{X}(\boldsymbol{i},\boldsymbol{j})$, dado que estas son sustituidas por la entrada de **DatoImage** (esta es un bus 8 bits para manejar imágenes con colores de 8 bits por canal de color), la cual es por donde se introducen los dados de la extracción de parches y éstos van directamente a la memoria interna. Observe que también adicionalmente ahora tiene tres entradas, la cuales son **Hmin**, **Vmin** y **flag**. Donde **flag** es una entrada de control la cual, teniendo un uno lógico, activa la operación de convolución siempre y cuando la entrada de **reset** este en un estado de cero lógico. Las entradas **Hmin** y **Vmin** se explican a continuación en la sección 4.4.2. Finalmente, las entradas con la letra $\boldsymbol{W}(\boldsymbol{i},\boldsymbol{j})$ es por donde se introduce el kernel y las salidas $\boldsymbol{F}(\boldsymbol{i},\boldsymbol{j})$ es por dónde se obtiene la imagen.

4.4.2 Memoria principal

Esta es la memoria interna del módulo de Conv2D, es decir el módulo de la Figura 4.13, como ya se hizo mención tiene una memoria interna, qué tiene arquitectura como la que se ilustrada en la Figura 4.14.

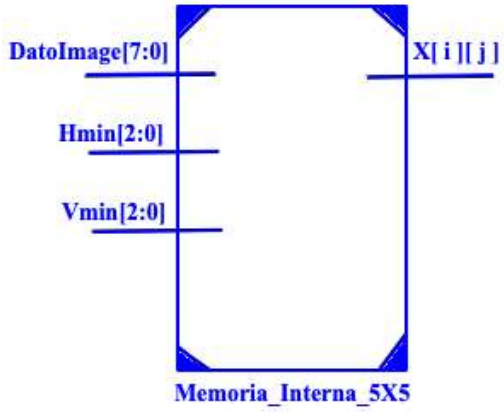


Figura 4.14, Arquitectura de la memoria principal.

En la Figura 4.14, se ilustra la arquitectura de la memoria que se encuentra internamente en el módulo Conv2D de la Figura 4.13. En la tabla 4.2, se ilustran más información de esta memoria.

Tabla 4.2, Especificaciones de la memoria principal.

PINES.	Bus.	Función.	Características.
DatoImage	8 bits.	Es el bus de entra donde se adquiere el dato del pixel extraído.	Dato sin signo.
Hmin	3 bits.	Dirección horizontal.	Dato valido 1 al 5.
Vmin	3 bits.	Dirección vertical.	Dato valido 1 al 5.
X[i][j]	8 bits.	Variable para leer o escribir en la memoria.	Dato sin signo.

Esta memoria permite almacenar una imagen de tamaño de 5×5 pixeles. Cabe recalcar que la **"salida X[i][j]"** entre comillas para resaltar qué no es una salida como tal, ya que como se encuentra internamente del módulo Conv2D de la Figura 4.13, la operación de convolución puede manipularla directamente, ya sea solo una región de memoria o bien toda la memoria.

4.4.3 Unidad de control

Como ya se planteó la unidad de control, es la encargada de decidir cuándo se almacenan los datos de la extracción de parches, así como en qué región de la memoria principal se almacenan, también decide cuándo se realiza la operación de convolución, además esta proporciona señales de control para leer una memoria que contenga la **imagen $\boldsymbol{x(i,j)}$,** así como proporciona una señal para escribir en una memoria donde se almacenara la **imagen filtrada** finalmente la unidad de control manipula un multiplexor, el cual nos permite extraer los datos de la operación de convolución de manera serial. El módulo de la unidad de control se ilustra en la Figura 4.15.

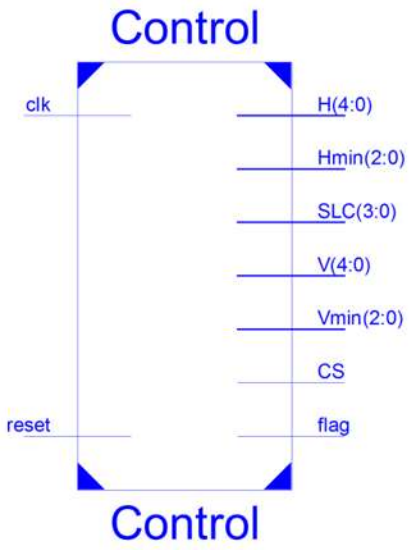


Figura 4.15, unidad de control.

En la Figura 4.15, se ilustra la unidad de control y en la tabla 4.3, se ilustran más información de esta unidad de control.

Tabla 4.3, Especificaciones de la unidad de control.

PINES.	Bus.	Función.	Características.
clk	1 bit.	Señal de reloj.	Sensible a flancos de subida.
reset	1 bit.	Señal para reiniciar la unidad.	1 = reinicia la toda unidad. 0 = la unidad trabaja.
H	5 bits.	Dirección horizontal, para controlar la memoria externa.	Datos validos del 1 al 25.
Hmin	3 bits.	Dirección horizontal, para controlar la memoria interna.	Datos validos del 1 al 5.
SLC	4 bits.	Señal que permite controlar un multiplexor.	Datos validos del 0 al 8.
V	5 bits.	Dirección vertical, para controlar la memoria externa.	Datos validos del 1 al 25.
Vmin	3 bits.	Dirección vertical, para controlar la memoria interna.	Datos validos del 1 al 5.
CS	1 bit.	Permite usarse como señal de reloj o habilitador de una memoria donde se almacenaran los datos obtenidos.	El periodo de esta señal es más lento que la señal de CLK.
flag	1 bit.	Señal que habilita o deshabilita la operación de convolución.	1 = habilita la operación de convolución. 0 = deshabilita la operación de convolución.

Antes de continuar hay que mencionar que la salida ***SLC*** permite controlar un multiplexor el cual ayudara a convertir los datos obtenidos por el módulo ***Conv2D*** de forma serial.

4.4.4 Obtención del procesador de convolución

En la Figura 4.16, se presenta la descripción final del procesador de convolución, al cual se le ha nombrado ***Ingrom*** y de aquí en adelante, nos refiramos a éste con ese nombre.

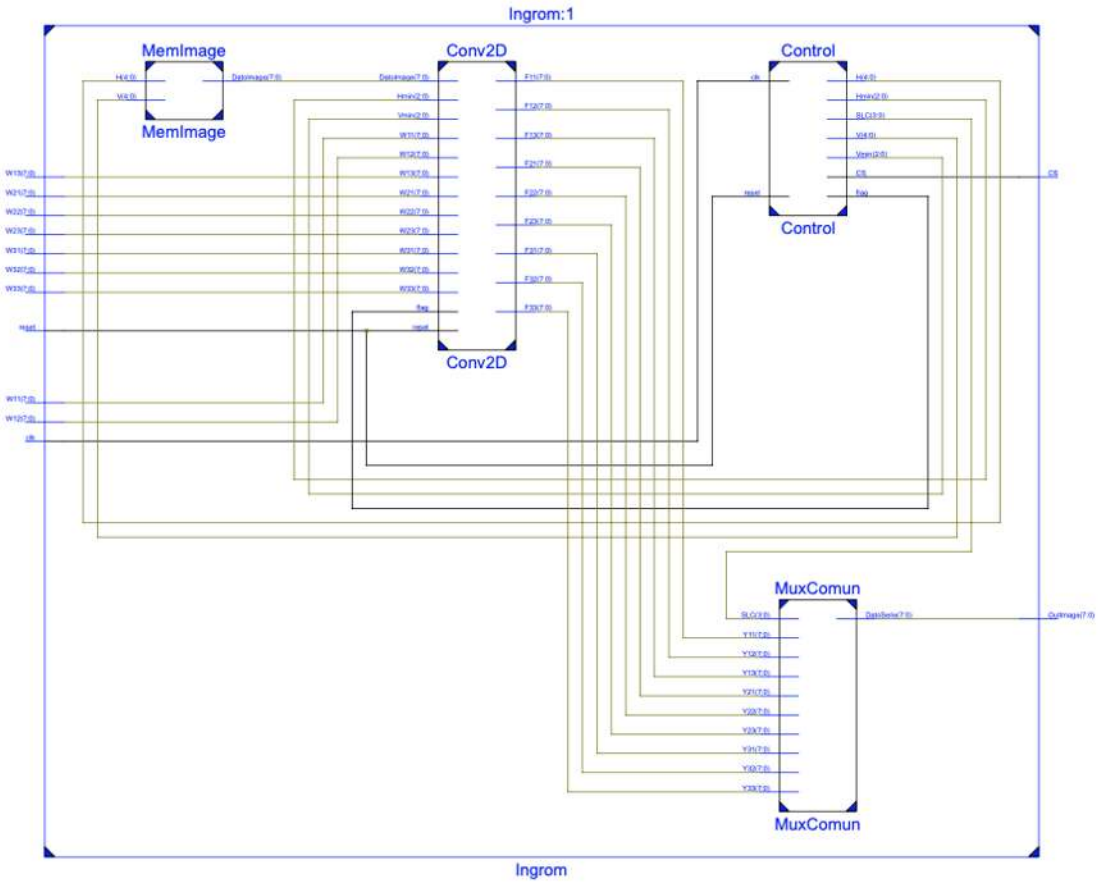


*Figura 4.16, Procesador de convolución **Ingrom**.*

Como se ilustra en la Figura 4.16, el procesador ***Ingrom***, tiene una arquitectura muy similar a la de la Figura 4.10, como se puede observar ***Ingrom*** cuenta con una unidad de control(***Control***), cuenta con la operacional de convolución(***Conv2D***) la cual internamente cuenta con la ***memoria principal*** y bloques de entrada y salida, además note que contiene una memoria llamada ***MemImage*** de tamaño de 25×25 y celdas de 8 bits, cabe recalcar que esta es la memoria es donde encuentra almacenada la **imagen** $\boldsymbol{x(i,j)}$ y solo se ha integrado con la finalidad de realizar las simulación que se presenta en la sección 4.4.5, es decir la memoria **MemImage** no es parte de la arquitectura y mucho menos es la memoria principal del procesador, recuerde que la memoria principal del procesador está internamente del módulo **Conv2D**.

Distinga que el módulo **MuxComun** no aparece en la arquitectura de la Figura 4.10, sin embargo, no quiere decir que éste no pertenezca a la arquitectura, ya que, si se recuerda que también la arquitectura cuenta con bloques de entradas y salidas, por lo que este módulo es una interfaz que permite extraer la imagen filtrada de forma serial, es decir este bloque pertenece a los bloques de salida. En la tabla 4,3, se ilustran más información de ***Ingrom***.

Tabla 4.3, Especificaciones de Ingrom.

PINES.	Bus.	Función.	Características.
clk	1 bit.	Señal de reloj.	Es sensible a flancos de sunida.
reset	1 bit.	Señal para reiniciar el procesador.	1 = reinicia la todo el procesador. 0 = el proceador trabaja.
W11 hasta W33	8 bits.	Son buses de entrada donde reciben el dato de cada pixel del Kernel.	El conjunto de estos buses recibe una Kernel de tamaño de $3x3$ pixeles y pueden ser datos con signo.
CS	1 bit.	Permite usarse como señal de reloj o habilitador de una memoria donde se almacenaran los datos obtenidos.	El periodo de esta señal es más lento que la señal de CLK.
OutImage	9×8 bits.	Se obtiene la imagen resultante o imagen filtrada, de forma serial.	Los datos obtenidos en estos buses pueden ser con signo o sin signo.

Capítulo 5

Diseño y descripción de HW de una interfaz de imagen

5.1 Introducción

En el capítulo anterior se diseñó y describió un procesador de convolución, el cual permite tratar el canal de una imagen de color o bien imágenes con un solo canal de color, sin embargo, nuestro interés es tratar con imágenes a color, es decir tratar con imágenes que están compuestas por varios canales. Por ello es que en este capítulo se expondrá el diseño y descripción de HW de una interfaz de imagen, la cual permitirá leer una imagen de tamaño de $105x105$ pixeles, con formato Bitmap (BMP) almacenada en una memoria.

5.2 Formato BMP

De acuerdo a lo mencionado en el capítulo 2.1.3, existen diversos formatos de compresión, los cuales favorecen el almacenamiento de imágenes digitales, dichos formatos tienen diversas características y como ya se sabe existen varios formatos, sin embargo, el formato de nuestro interés es el formato BMP de 16 bits por pixel.

BMP es el formato de imagen nativo en los sistemas operativos Microsoft Windows. Admite imágenes con 1, 4, 8, 16, 24 y 32 bits por píxel, La orden de los datos en el formato BMP de Windows **se almacenan primero con los bytes menos significativos** es decir usa el sistema ***little-endian***, además el formato BMP usa un esquema de color **RGB**. Los datos almacenados en formato BMP constan completamente de bytes completos, por lo que el orden de las cadenas de bits no es un problema. [44].

La estructura del archivo BMP es muy simple y se muestra en la tabla 5.1 [45]. Al conocer la estructura se puede determinar como el archivo BMP está almacenado en la memoria y así extraer los datos de la imagen y que dichos datos puedan ser manipulados.

Tabla 5.1, Estructura del archivo BMP.

Nombre del campo	Tamaño en bytes	Bytes	Información
bfType	2	0, 1	Contiene los caracteres BM que identifica el tipo de archivo BMP.
bfSize	4	2, 3, 4, 5	Tamaño de archivo.
bfReserved1	2	6, 7	Reservado.
bfReserved2	2	8, 9	Reservado.
bfOffBits	4	10, 11, 12, 13	**Inicio de los datos de imagen.**
biSize	4	14, 15, 16, 17	Tamaño de la cabecera del bitmap.
biWidth	4	18, 19, 20, 21	**Ancho de la imagen.**
biHeight	4	22, 23, 24, 25	**Altura de la imagen.**
biPlanes	2	26, 27	Numero de planos.
biBitCount	2	28, 29	Bits por pixel.
biCompression	4	30, 31, 32, 33	Tipo de 4 compresión)BI_RGB = 0, BI_RLE8 = 1, BI_RLE4 = 2, o BI_BITFIELDS = 3.
biSizeImage	4	34, 35, 36, 37	Tamaño de la imagen.
biXPelsPerMeter	4	38, 39, 40, 41	Resolución horizontal.
biYPelsPerMeter	4	42, 43, 44, 45	Resolución vertical.
biClrUsed	4	46, 47, 48, 49	Tamaño de tabla de color
biClrImportant	4	50, 51, 52, 53	Número de colores significativos

5.3 Diseño de la interfaz de imagen

De acuerdo con lo mencionado anteriormente, el sistema que se diseño tiene la estructura que se ilustra en el diagrama en bloques de la Figura 5.1.

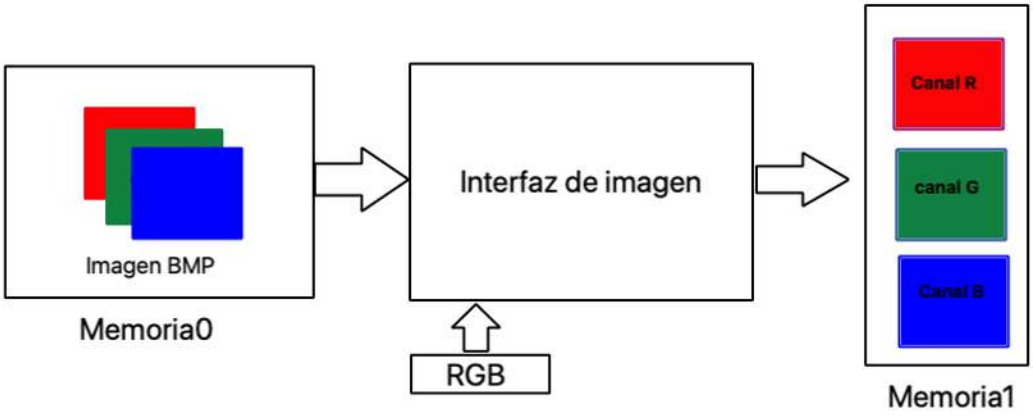


Figura 5.1, Diagrama a bloques de la interfaz de imagen.

Este diseño se basa principalmente en el bloque interfaz de imagen de la Figura 5.1, el cual como se puede apreciar toma especialmente una instrucción ilustrada por el bloque **RGB**, éste le indica al bloque **interfaz de imagen**, el canal de la imagen que se extraerá.

La imagen con el formato BMP y sistema RGB, como se puede ver se encuentra almacenada en una memoria, la cual se ilustra por el bloque con el nombre **Memoria0**. El bloque **interfaz de imagen** tiene específicamente considerada la tabla 5.1, la cual, con la ayuda de ésta, determina donde se encuentran almacenados los datos de la imagen en la **Memoria0**, así con estos y la instrucción del bloque **RGB**, así extrae el canal de la imagen y se almacena en una segunda memoria. Como se observa la segunda memoria esta ilustrada por el bloque **Memoria1** y este contiene uno de los tres canales RGB; observe que el bloque **Memoria1** ilustra internamente los tres canales para representar que el canal extraído puede ser uno de los tres casos. Es importante hacer énfasis en que los bloques de memoria ilustrados como **Memoria0** y **Memoria1**, pueden ser memorias separadas o regiones de una sola memoria.

5.4 Descripción de HW de la interfaz de imagen

La descripción de HW de la interfaz de imagen se realizó a partir del diseño de la Figura 5.1, y se realizó por partes, para efectuar la simulación expuesta en capítulo 5.9. Dichas etapas se exponen a continuación en siguientes secciones.

5.4.1 Descripción de una memoria ROM que almacenara la imagen con formato BMP

Primeramente, se describió una memoria ROM, la cual es representada por el bloque **Memoria0**, es decir se describió la memoria en la cual se encontrará almacenada la imagen con formato BMP, la arquitectura de la memoria se ilustra en la Figura 5.2.

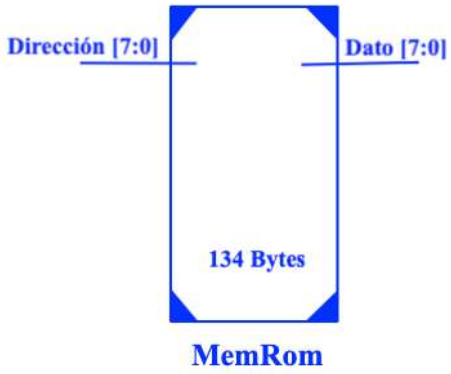


Figura 5.2, Arquitectura de la memoria que almacena la imagen con formato BMP.

En la Figura 5.2, se ilustra la arquitectura de la memoria que contendrá la imagen BMP, como se puede ver esta tiene un tamaño de 134 Bytes los suficientes para almacenar una imagen de tamaño de 5×5 pixeles, como ya se mencionó en el anterior capítulo nuestro interés es manipular imágenes de tamaño 105×105 sin embargo el hecho de que se estén manejando imágenes más pequeñas a las de nuestro interés, tiene la finalidad de realizar una explicación más franca y poder realizar la simulación. En la tabla 5.2, se ilustra más información de ***MemRom***.

Tabla 5.2, Especificaciones de MemRom.

PINES.	Bus.	Función.	Características.
Dirección	8 bits.	Dirección de la memoria.	Dato valido 1 al 134.
Dato	8 bits.	Datos de la imagen.	Datos sin signo.
NC	8 bits	Almacenamiento ROM	134 celdas de 8 bits

Cabe recalcar que esta memoria ROM lee directamente datos que están descritos un archivo de texto con el formato **dat**, este archivo permite almacenar información en hexadecimal, es decir el archivo contendrá los datos en hexadecimal de la imagen con formato BMP que se desea manipular. El archivo **dat** se lee directamente con un comando, el cual le indica al sintetizador del *ISE* de *Xilinx* que extraiga los datos de un archivo **dat** y los almacene en una memoria descrita. El uso de este comando se ilustra en la Figura 5.3.

```
$readmemh("ubicación_del_archivo\\Nombre_del_arhivo.dat", memoria)
```

Figura 5.3. Estructura del comando $readmemh

Como se ilustra en la Figura 5.3, el comando se llama $readmeh y esta resaltado de color azul, observe que entre el paréntesis recibe varios argumentos. Dentro comillas dobles reciben la ubicación y el nombre del archivo con formato **dat** y el argumento que está separado con una como recibe el nombre de la memoria a la que se le quiere escribir la información del archivo con formato **dat**.

5.4.2 Descripción de Memoria que almacenara el canal extraído

La memoria que almacenara el canal de la imagen esta descrita con la arquitectura que se ilustra en la Figura 5.4.

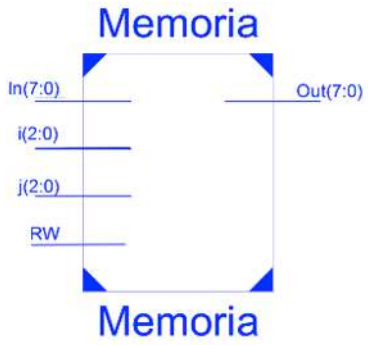


Figura 5.4, Arquitectura que almacenará el canal extraído de la imagen.

La memoria ilustrada en la Figura 5.4, tiene un tamaño de 5×5 y celdas de 8 bits, con la finalidad de almacenar el canal extraído de una imagen de tamaño 5×5, en la tabla 5.3, se ilustra la información del módulo ***Memoria.***

Tabla 5.3, Especificaciones del modulo Memoria.

PINES.	Bus.	Función.	Características.
In	8 bits.	Este bus recibe los datos.	Sin signo.
i	3 bits.	Dirección vertical.	Dato valido del 1 al 5.
j	3 bits.	Dirección horizontal.	Dato valido del 1 al 5.
Out	8 bits.	Este bus obtiene los datos.	Sin signo.

La memoria ilustrada en la Figura 5.4, es una memoria que puede ser compartida con el procesador ***Ingrom*** ya que como ya se conoce esta contendrá el canal de la imagen extraído y ***Ingrom*** puede manipularlo directamente.

5.4.3 Descripción de la interfaz de imagen

La interfaz tiene como núcleo una unidad de control, el cual decide cuándo leer la memoria que contiene la imagen y también cuándo escribir en la memoria donde se almacenara el canal extraído. De acuerdo con lo mencionado la unidad tiene establecido los números de los Bytes que permiten conocer el Byte donde **inician los datos de imagen**, también tiene establecido los Bytes para conocer el **ancho de la imagen** y la **altura de la imagen**, los números de estos Bytes se ilustran en la tabla 5.1. La arquitectura de la unidad se ilustra en la Figura 5.5.

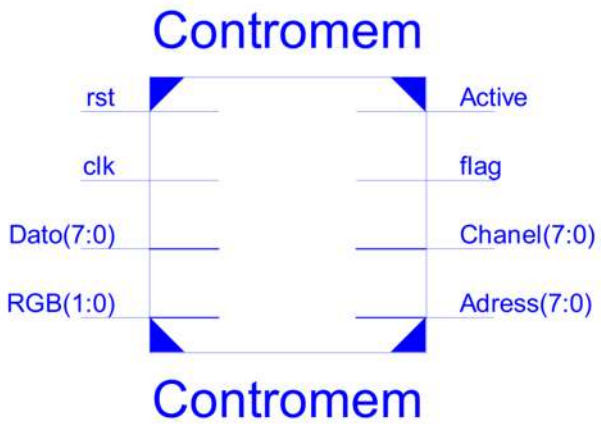


Figura 5.5, Arquitectura de la interfaz de imagen.

Como se puede ver en la Figura 5.5, tiene cierta similitud con la Figura 5.1. En la tabla 5.3 se ilustra más información de ***Contromem.***

Tabla 5.3, Especificaciones de Contromem.

PINES.	Bus.	Función.	Características.
clk	1 bit.	Señal de reloj.	Sensible a flancos de subida.
rst	1 bit.	Señal para reiniciar la interfaz.	1 = reinicia la interfaz. 0 = la interfaz trabaja.
Dato	8 bits.	Recibe los datos extraídos de una memoria donde esta almacenada la imagen BMP.	Datos sin signo.
RGB	2 bits.	Selecciona el canal a extraer.	Datos validos: 00'b=RED. 01'b=GREN. 10'b=BLUE.
Active	1 bit.	Es la señal de reloj para la unidad control0.	Es solo una señal.
flag	1 bit.	Es una señal de control la cual permite leer o escribir a la memoria en donde se almacenará el canal extraído y con la ayuda de un multiplexor decide que unidad de control se utiliza.	Es solo una señal.
Chanel	8 bits.	Es por donde se adquieren los datos del canal extraído.	Datos sin signo.
Adress	8 bits.	Es la encargada de manipular la dirección de la memoria que contiene la imagen BMP.	Dirección valida del 1 al 134.

5.4.4 Unidades de control de la memoria donde se almacena la extracción del canal de la imagen

La memoria donde se almacena la extracción cuenta con dos unidades de control, dado que la unidad **contromem** ilustrada en la Figura 5.5, obtiene el canal de la imagen, primeramente, con la última fila de la imagen y por último la primer fila de la imagen, es decir reacomoda la extracción del canal para que tenga un posición natural, en la memoria. La arquitectura se ilustra en la Figura 5.6.

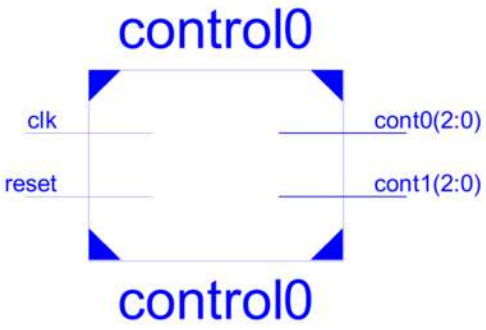


Figura 5.6, Especificaciones de control0.

El módulo de la Figura 5.6, está compuesto por dos contadores internamente los cuales son descendentes. En la tabla 5.4 se ilustra más información de ***control0.***

Tabla 5.4, Especificaciones de control0.

PINES.	Bus.	Función.	Características.
clk	1 bit.	Señal de reloj la cual es proporcionada por la señal Active.	Es sensible a flancos de subida.
reset	1 bit.	Reinicia la unidad.	1 = reinicia la unidad. 0 = la unidad trabaja.
cont0	3 bits.	Salida del primer contador.	La salida va del 5 al 1.
cont1	3 bits.	Salida del segundo contador.	La salida va del 5 al 1.

La segunda unidad permite extraer la imagen ya con una posición correcta; es decir extrae la primer fila hasta llegar a la última fila de la imagen con una posición natural. La arquitectura de la unidad a la que nos referimos se ilustra en la Figura 5.7.

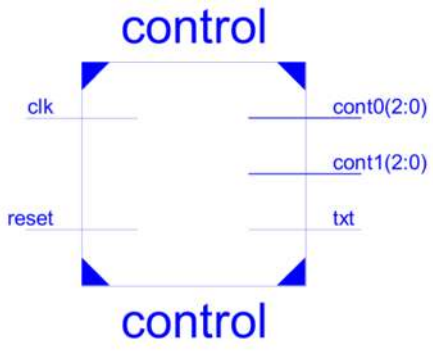


Figura 5.7, Arquitectura del control.

La arquitectura del módulo ilustrado en la Figura 5.7, tiene contadores internos los cuales de igual manera permiten tener un valor máximo de cinco en las salidas **cont0** y **cont1** para

controlar memoria donde se almacena la extracción. Cabe recalcar que los contadores internos es un contador ascendente y uno descendiente. En la tabla 5.5 se ilustra más información de ***control.***

Tabla 5.5, Especificaciones de control.

PINES.	Bus.	Función.	Características.
clk	1 bit.	Señal de reloj la cual es proporcionada por la señal Active.	Es sensible a flancos de subida.
reset	1 bit.	Reinicia la unidad.	1 = reinicia la unidad. 0 = la unidad trabaja.
cont0	3 bits.	Salida del primer contador.	La salida va del 5 al 1.
cont1	3 bits.	Salida del segundo contador.	La salida va del 1 al 5.
txt	1 bit.	Es una señal que indica el intervalo de la primera lectura a la memoria.	Si la señal tiene el valor de: 1 = Se esta escribiendo en la memoria. 0 = Se termino de escribir en la memoria y el canal de la imagen extraído esta listo.

Estos controladores no pueden ser conectados directamente a la memoria donde se almacena la extracción ya que como se habló uno permite escribir y otro leer. Para solucionar este problema se usó un multiplexor el cual está controlado directamente con la salida **flag** de la unidad ilustrado en la Figura 5.5, se sabe que esta salida indica, cuándo se lee y se escribe en la memoria que almacena la extracción. La arquitectura se ilustra en la Figura 5.8.

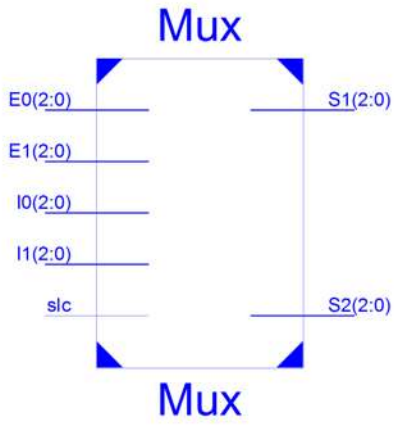


Figura 5.8, arquitectura de Mux.

En la Figura 5.8, se ilustra la arquitectura de **Mux**, además en la tabla 5.6, se indica más información de este.

Tabla 5.6, Especificaciones del Mux.

PINES.	Bus.	Función.	Características.
E0	3 bits.	Recibe la salida **cont0** del ***control0*** ilustrado en la Figura 5.6.	Solo es un bus entrada.
E1	3 bits.	Recibe el bus de salida llamado **cont1** del ***control0*** ilustrado en la Figura 5.6.	Solo es un bus entrada.
I0	3 bits.	Recibe el bus de salida llamado **cont0** del ***control*** ilustrado en la Figura 5.7.	Solo es un bus entrada.
I1	3 bits.	Recibe el bus de salida llamdo **cont1** del ***control*** ilustrado en la Figura 5.7.	Solo es un bus entrada.
SLC	1 bit.	Es la entrada qué recibe la salida **flag** de ***controlmem.***	Si SLC=0: S1=I0; S2=I1(se lee la memoria). Si SLC=1: S1=E0; S2=E1(se escribe en la memoria).
S1	3 bits.	Es la salida de multiplexor.	Solo es un bus de salida.
S2	3 bits.	Es la salida de multiplexor.	Solo es un bus de salida.

5.4.5 Sistema final de la interfaz de imagen.

El sistema final que contiene el núcleo de la unidad de interfaz de imagen y conjunto de las memorias y controles de memoria se ilustra en la Figura 5.9.

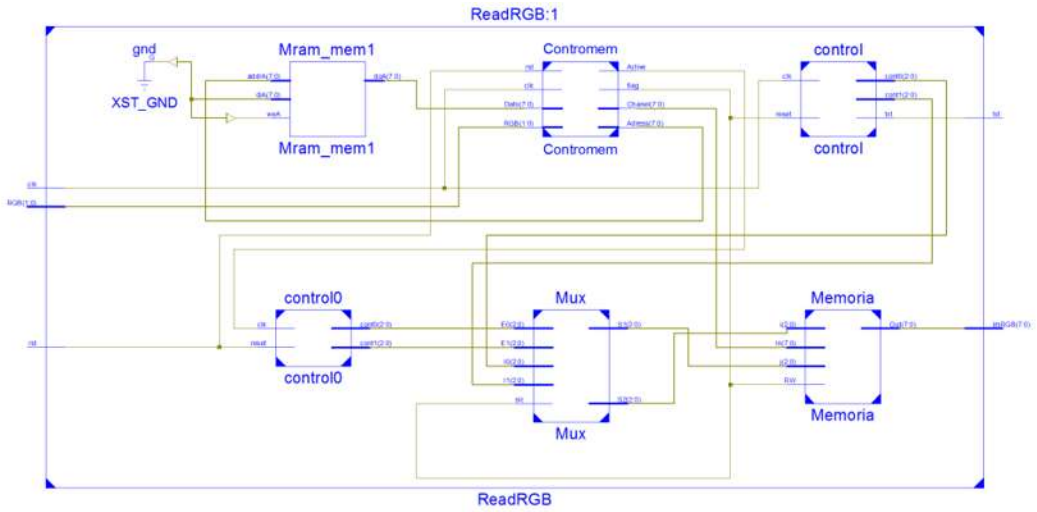


Figura 5.9, Arquitectura de la integración de todos los módulos que componen la interfaz de imagen completa.

En la Figura 5.9, se ilustra como están interconectados todos los módulos explicados anteriormente, en la Figura 5.10, se ilustra el módulo obtenido finalmente.

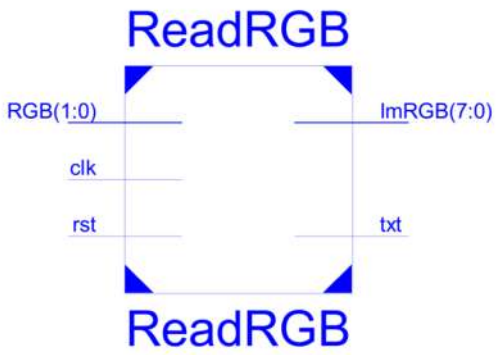


Figura 5.2, módulo ReadRGB.

El módulo de la Figura 5.10, es el conjunto de todos los bloques ilustrados en la Figura 5.1. En la tabla 5.7, se ilustra más información de este módulo.

Tabla 5.7, Especificaciones de ReadRGB.

PINES.	Bus.	Función.	Características.
clk	1 bit.	Señal de reloj.	Sensible a flancos de subida.
rst	1 bit.	Reinicia la unidad.	1 = reinicia la unidad. 0 = la unidad trabaja.
RGB	2 bits.	Selecciona el canal a extraer.	Datos validos: 00’b = RED.

			01’b = GREN. 10’b = BLUE.
ImRGB	8 bits.	Es el bus de salida por donde se obtiene los datos del canal de la imagen de forma serial.	Son pixeles de canal extraído.
txt	1 bit.	Indicador de la primer lectura de la memoria que almacena la imagen.	Es solo una señal.

Capítulo 6

Diseño y descripción de HW de un procesador de imágenes

6.1 Introducción

En los capítulos anteriores se mostró el diseñado y descripción de HW de dos dispositivos muy importantes para el PDI, el cual como se conoce es un tema que aborda a las SRCNN y en presente capitulo se aborda el diseño y la descripción de HW de un procesador de imágenes el cual es una herramienta fundamental para las SRCNN.

6.2 Obtención de un procesador de imagen

En el capítulo anterior se mostro el diseño y descripción de HW de una interfaz de imagen que permite leer imágenes con el formato BMP, con la finalidad de que la lectura pueda ser manipulada, por el procesador de convolución, previsto en el capítulo cuatro. La unión tanto del procesador de convolución y la interfaz de imagen radican en un procesador

de imagen (ISP, por sus siglas en inglés). El proceso de la unión se menciona en las siguientes secciones.

6.2.1 Modificación del procesador *Ingrom*

Como primer paso, se le realizaron modificaciones al procesador ***Ingrom***, ya que como se ilustra en la Figura 4.23, este cuenta con una memoria ilustrada por el módulo **MemImage**, la cual almacena un canal de una imagen a procesar. Recordemos que esta memoria no constituye en la arquitectura del procesador de convolución y debe estar fuera de ***Ingrom***. El procesador modificado ***Ingrom*** obtenido se ilustra en la Figura 6.1.

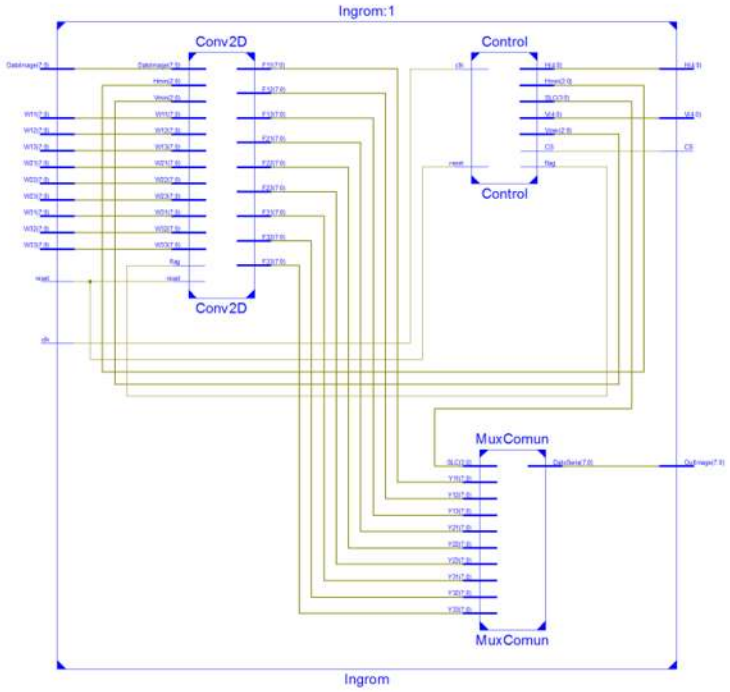


*Figura 6.1, Modificación de **Ingrom**.*

Como se puede apreciar en la Figura 6.1, ***Ingrom*** no contiene la memoria ***MemImage***, observe que adicionalmente se le agrego un bus entrada de 8 bits llamado ***DatoImage***, la cual es por donde recibirá de manera serial el canal de la imagen a procesar, también se le agregó dos buses de salida de 4 bits indicados como ***H*** y ***V***, los cuales permiten controlar una memoria externa que contenga el canal de la imagen. Dicho de otra manera, la salida

OutImage permiten extraer los datos de una memoria compatible y dichos datos son recibidos por el bus de entrada ***DatoImage***.

6.2.2 Modificación del HW del módulo ReadRGB

Posteriormente se modificó el módulo **ReadRGB**, ya que como podemos observar en la Figura 5.9, éste cuenta con una memoria ilustrada por el módulo **Memoria**, la cual es una memoria compatible donde se almacenan los datos del canal extraído, de la imagen con formato ***BMP***, es decir esta memoria es una memoria externa, la cual no es parte de la arquitectura del módulo **ReadRGB**, por lo que tuvo que extraerse. También se extrajo el módulo ilustrado con el nombre **Mux** ya que es una pequeña interfaz que permite conectar unidades de control que pueden leer y escribir de forma descendente o accedente en la memoria extraída. Finalmente se extrajo el módulo ilustrado con el nombre de **Control**, ya que es una unidad de control que está conectada al módulo **Mux** para leer la memoria de forma descendente. Recordemos que estos módulos solo se agregaron para realizar la simulación del capítulo 5.9, y por ello no forman parte de la arquitectura del módulo **ReadRGB**. El módulo modificado se ilustra en la Figura 6.2.

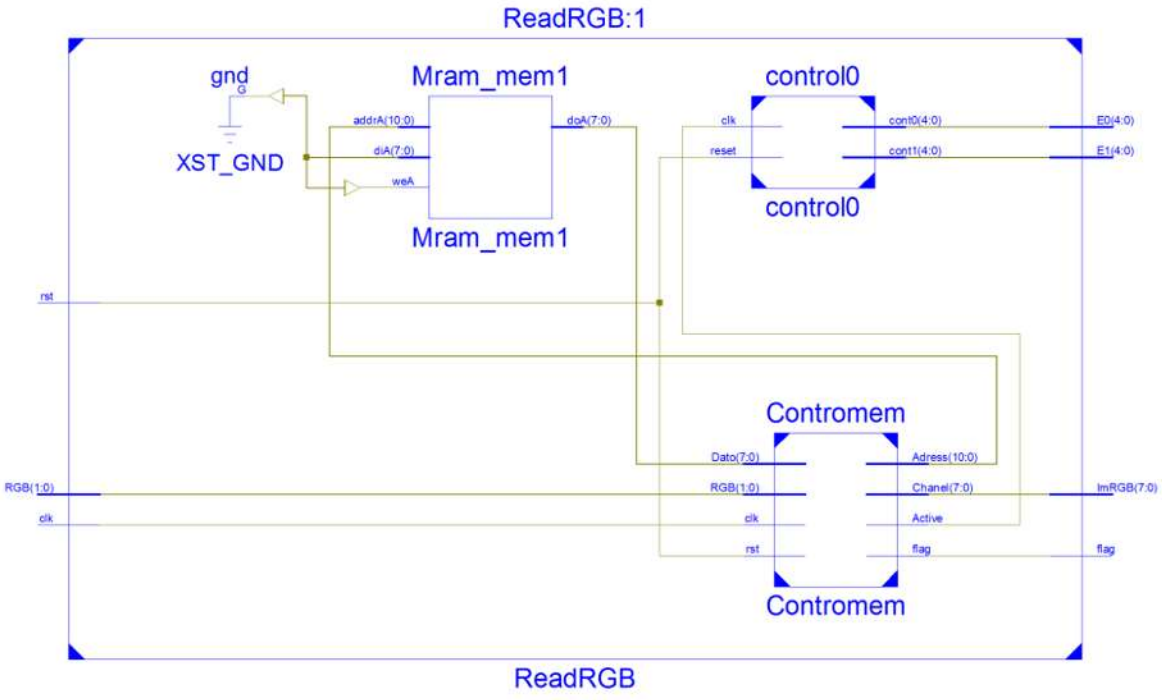


Figura 6.2. Modificación de módulo ReadRGB.

Como se observa en la Figura 6.2, la interfaz de imagen **ReadRGB** no contiene los módulos **Memoria**, **Mux** y **Control**, observe que adicionalmente se agregaron dos buses de salida de 5 bits los cuales están indicados con los nombres de **E0** y **E1**, éstas salidas junto con la salida **flag** deciden cuándo escribir los datos del canal de la imagen, en una memoria externa compatible, cabe recalcar que **flag** indica si lee o escribe dicha memoria y además le indica a ***Ingrom*** cuándo puede ejecutarse.

6.2.3 ISP PI

Una vez realizado las modificaciones de los módulos **Ingrom** y **ReadRGB** se describió el procesador de imagen, la arquitectura de la descripción de **HW** del **ISP** se ilustra en la Figura 6.3.

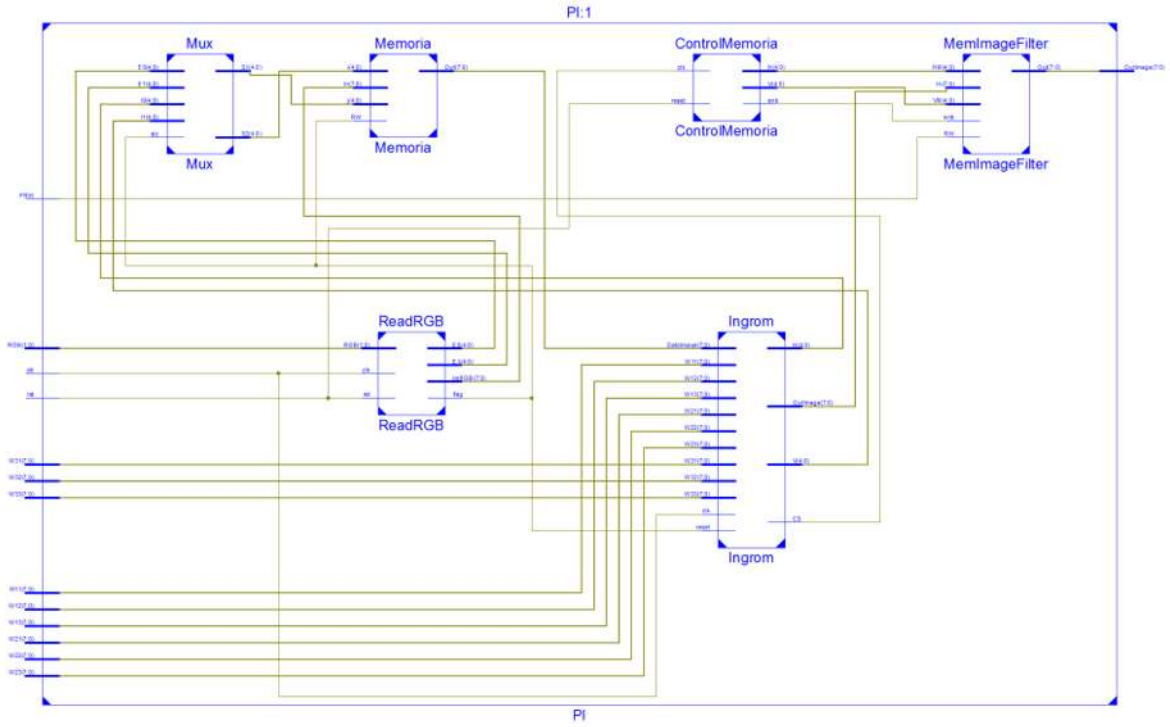


Figura 6.3, Arquitectura del procesador de imagen.

En la Figura 6.3, se ilustra la arquitectura del ISP, sin embargo, para su explicación se usará el diagrama a bloques de la Figura 6.4.

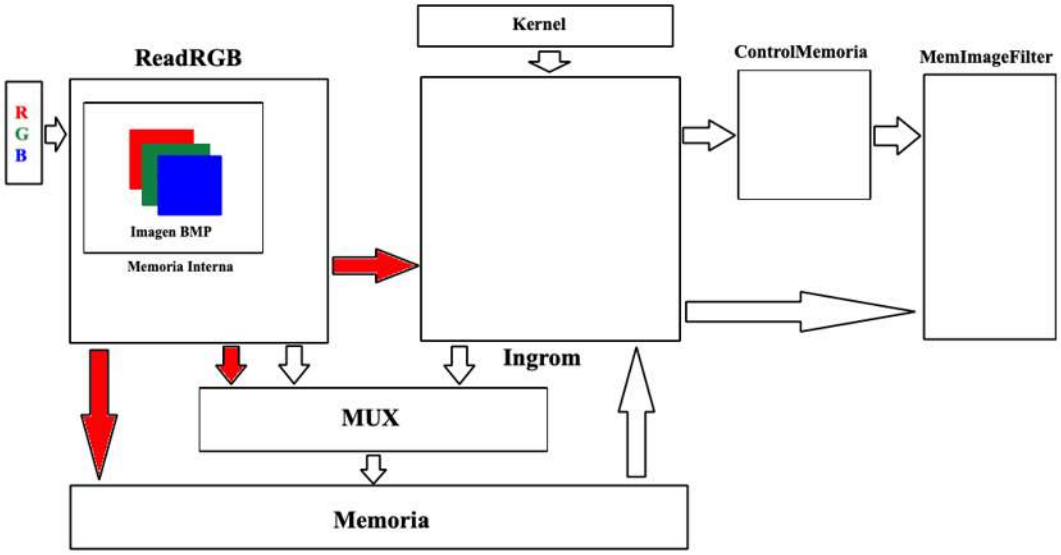


Figura 6.4, Diagrama a bloques de la arquitectura del procesador de Imagen.

En la Figura 6.4, se ilustra el diagrama a bloques de la arquitectura del procesador de imagen, principalmente se puede observar que el bloque con el nombre de RGB, representa la entrada la cual es por donde se le da la instrucción a **ReadRGB**, para extraer el canal de la imagen ***BMP***, que como se sabe esta se encuentra almacenada en una memoria interna de módulo **ReadRGB**.

El módulo **ReadRGB** apunta hacia los bloques: **Memoria**, **Mux** y ***Ingrom***; con flechas de color rojo, las cuales representan la señal **flag**, es decir, esta le indica a la memoria cuándo escribir y leer y a su vez le indica al multiplexor ilustrado por el bloque **Mux**, que unidad de control permite controlar la lectura o escritura de la memoria, ilustrada por el bloque **Memoria**; la señal **Flag** también le indica a ***Ingrom*** cuándo puede operar.

Como ya se mencionó el bloque **Mux** es una pequeña interfaz que permite elegir entre realizar escritura de **ReadRGB** o una lectura de la memoria, observe que la lectura está controlada por el bloque ***Ingrom***, el cual puede obtener los datos entregados por el bloque Memoria, realiza la **convolución** con el **kernel** introducido en la entrada ilustrada por el bloque **Kernel** y el resultado lo envía al bloque ilustrado como **MemFilter**, dichos datos son almacenados en el bloque **MemImageFilter** según la dirección proporcionada por el bloque **ControlMemoria**.

El PSI que se obtuvo como tal es el que se ilustra en la Figura 6.5, el cual se le nombro como ***PI***.

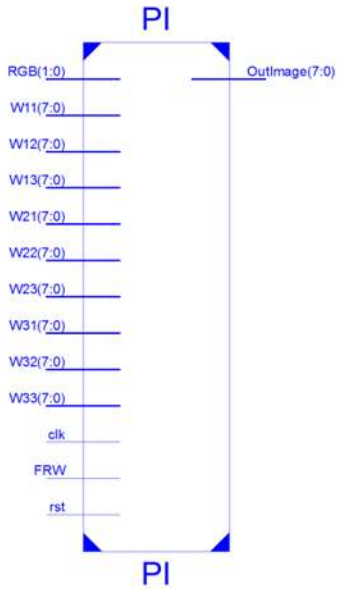


Figura 6.5, ISP PI.

En la Figura 6.5, se ilustra el ISP ***PI***, en la tabla 6.1, se ilustra más información de ***PI***.

Tabla 6.1, Especificaciones de ***PI***.

PINES.	Bus.	Función.	Características.
clk	1 bit.	Señal de reloj.	Sensible a flancos de subida.
rst	1 bit.	Reinicia el procesador.	1 = reinicia el procesador. 0 = el procesador trabaja.
FRW	1 bit.	Permite indicarle a la memoria **MemFilter**, si se lee o se le escribe.	1 = escribe en la memoria. 0 = lee la memoria.
RGB	2 bits.	Selecciona el canal a extraer.	Datos validos: 00'b = RED. 01'b = GREN. 10'b = BLUE.
W11 hasta W33	8 bits.	Estos buses reciben los pixeles del kernel.	Los datos recibidos pueden ser con signo.
OutImage	8 bits.	Permite obtener los datos de la lectura de la memoria **MemFilter.**	Los datos obtenidos son pixeles.

Capítulo 7

Diseño y descripción de una capa de convolución para el algoritmo SRCNN:

7.1 Introducción

Hasta el momento se ha obtenido una herramienta fundamental para para implementar un algoritmo de SR en una FPGA, de tal forma que los módulos descritos en HW cumplen con las necesidades de poder realizar PDI, sin embrago no están enfocados directamente para la implementación del algoritmo de SR propuesto Chao Dong et al [46] y presentado en el capítulo tres, por ello es que en esta sección se expondrá el diseño y descripción de HW para adquirir una arquitectura de HW que cubra parcialmente la necesidad del algoritmo de SR propuesto.

7.2 Estudio del algoritmo SRCNN

Para conocer realmente las necesidades del algoritmo se estudió la implementación de la SRCNN en Matlab [47], dicha implementación fue realizada por Chao Dong et al y parten de haber adquirido y realizar entrenamiento de la SRCNN es decir se conoce que para poder realizar dicha implementación obtuvieron primeramente los pesos y sesgos a partir del entrenamiento de la SRCNN. En las figuras 7.1 y 7.2, se ilustra un diagrama de flujo de dicha implementación.

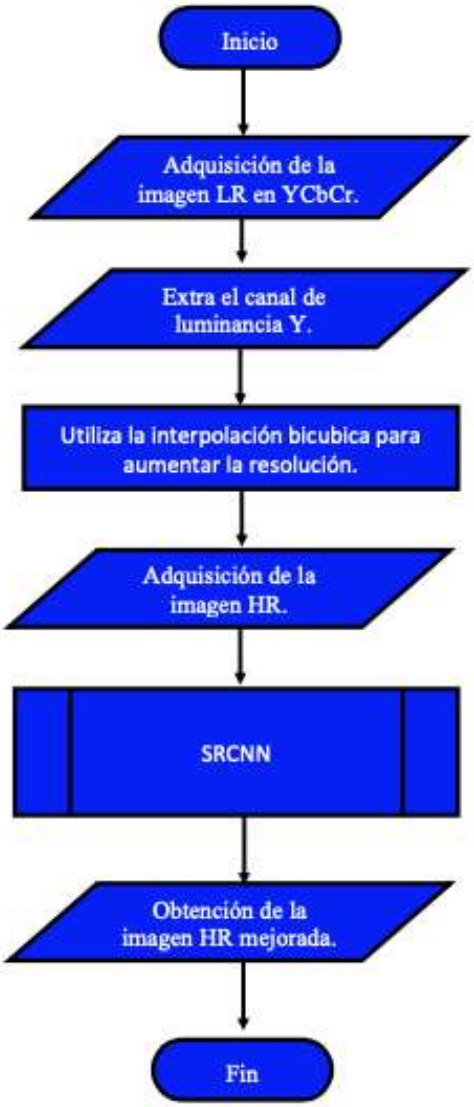


Figura 7.1, Primer diagrama de flujo de la SRCN implementada en Matlab.

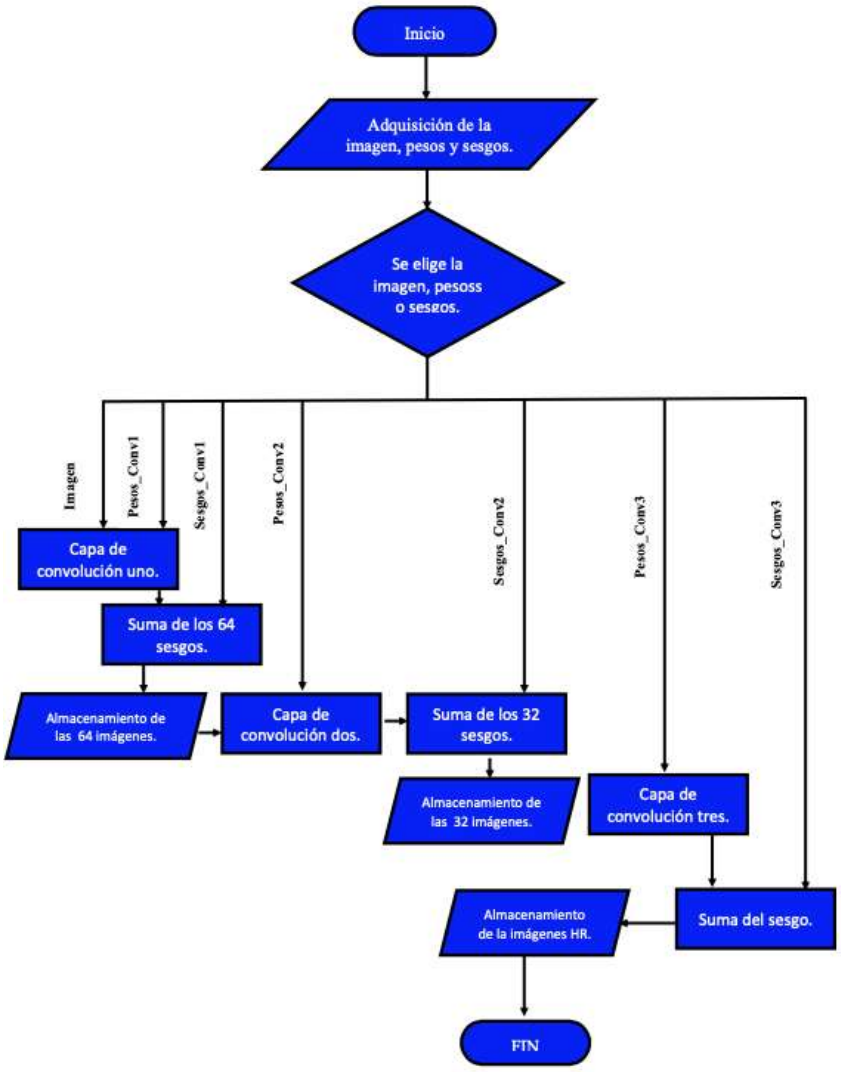


Figura 7.2, Segundo Diagrama de flujo de la implementación de la SRCNN en Matlab.

El diseño y descripción de HW de la arquitectura obtenida están enfocados en el proceso SRCNN ilustrado en la Figura 7.1, específicamente al proceso de la capa de convolución uno ilustrada en la Figura 7.2.

7.3 Adquisición de la imagen, pesos y sesgos:

7.3.1 Memoria para almacenar la imagen LR:

Principalmente se obtuvo una memoria ROM donde se encuentra almacenado el canal de iluminancia de una imagen YCbCr a la cual se le aplico la interpolación bicúbica, es decir la memoria ROM contiene la imagen HR.

Dicha memoria ROM fue descrita de la misma forma que la memoria presentada en sección 5.4.1, además de que se agregó un codificador para que el procesador ***Ingrom*** pueda manipular directamente dicha memoria. La arquitectura de esta memoria se ilustra en la Figura 7.3.

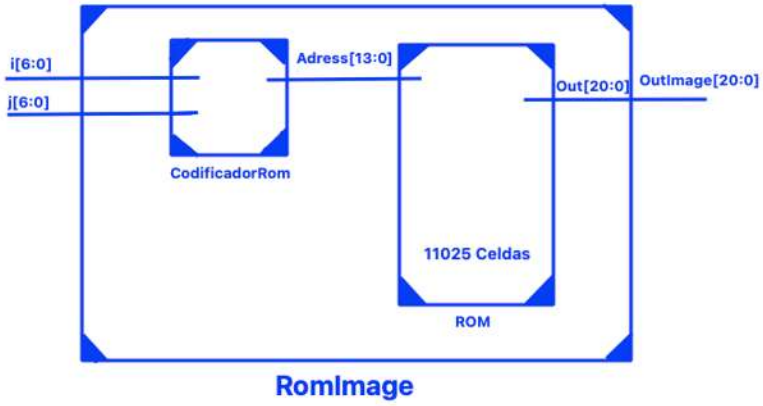


Figura 7.3, Memoria ROM para la imagen HR.

Esta memoria ROM permite almacenar una imagen de 105×105 pixeles, en la tabla 7.1, se ilustra más información de dicha memoria.

Tabla 7.1, Especificaciones de la memoria ROM para la imagen HR.

Pines	Bus	Características
i	7 bits	Dato valido 1 a 105.
j	7 bits	Dato valido 1 a 105.
Outimage	21 bits	Dato con signo.
		Rom con 11025 celdas de 21 bits.

Como se puede observar las características de la memoria ROM ilustradas en la tabla 7.1, se puede deducir que la memoria esta sobrepasada para almacenar imágenes con esquema de

color de 8bits, sin embargo, al estudiar la implementación del algoritmo en Matlab se observó que para la imagen de LR se le realiza una normalización para que los pixeles tengan valores entre cero y uno, es decir los pixeles de la imagen tienen asignado números reales. Además, se puede observar que la memoria maneja datos con signo, aunque la imagen normalizada no cuenta con pixeles con valores con signo, sin embargo, los pesos y los sesgos sí, cabe recalcar qué los sesgos también tienen valores reales y es por ello es qué las celdas son de 21 bits y además se usó la codificación de punto fijo para almacenar los datos de la imagen, sesgos y pesos.

La codificación de punto fijo que se usó consiste de la siguiente manera:

Como se sabe al tener veinte bits se puede representar un valor máximo $2^{20} = 1{,}048{,}575$ así que la codificación de los datos se ilustra en la imagen 7.4.

Figura 7.4, Codificación de punto fijo.

Como se puede apreciar en la Figura 7.4, representa un número máximo el cual se pretende representar, es decir 999,999 como podemos ver si es representable por los 20 bits sin embargo se puede apreciar que este valor máximo esta divido en dos secciones una de color azul la cual representa dos dígitos para los valores enteros y la sección de color rojo representa cuatro dígitos para los valores decimales de esta forma es como se codificaron lo datos para representa números reales, adicionalmente se aprecia una celda que esta resaltada de color morado, la cual representa el bit de signo y con ella se completan las celdas de 21 bits. Nuevamente cabe recalcar que las memorias diseñadas a partir de aquí cuentan con esta codificación para almacenar la imagen de LR, pesos, sesgos y además las imágenes resultantes de cada proceso de la implementación de la SRCNN.

Como se puede ver en la Figura 7.3, esta arquitectura de memoria contiene un codificador cuya arquitectura es muy simple y se ilustra en la Figura 7.5.

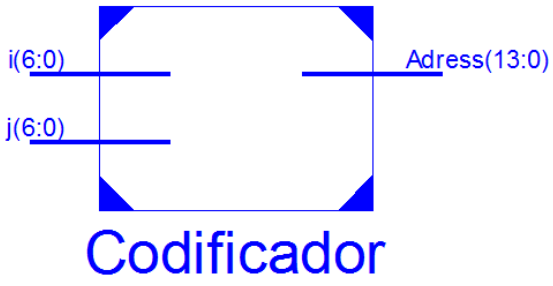


Figura 7.5, Arquitectura del Codificador.

Este codificador permite convertir dos direcciones en una sola de tal modo que se pueda tratar una memoria lineal como una memoria matricial. En la tabla 7.2, se ilustras más información de este codificador.

Tabla 7.2, Especificaciones del Codificador.

PINES.	Bus.	Función .	Características.
i	7 bits.	Dirección vertical.	Dato valido 1 a 105.
j	7 bits.	Dirección horizontal.	Dato valido 1 a 105.
Adress.	14 bits.	Se obtiene las direcciones i, j en una sola dirección.	Dato obtenido con un rango de 1 a 11025.

7.3.2 Memoria para almacenar los pesos:

Los pesos y los sesgos son los elementos de la SRCNN que son obtenidos mediante el entrenamiento y al estudiar la implementación del algoritmo en Matlab, Chao Dong et al [47] proporcionan varios modelos de entrenamiento el modelo en el que está enfocado la descripción del HW es en el modelo 9-5-5 x3.mat [47], este modelo tiene las características ilustradas en la tabla 7.3.

Tabla 7.3, Características del modelo 9-5-5 x3.mat.

Variable	Tamaño	Cantidad	Función	Aplicación
Pesos_Conv1	9×9	64	Kernels	Para la primer capa de convolución.
Pesos_Conv2	5×5	2048	Kernels	Para la segunda capa de convolución.
Pesos_Conv3	5×5	32	Kernels	Para la tercera capa de convolución.
Sesgos_Conv1	64×1	1	Sumando	Para la primer capa de convolución.
Sesgos_Conv2	32×1	1	Sumando	Para la segunda capa de convolución.
Sesgos_Conv3	1	1	Sumando	Para la tercera capa de convolución.

Como se aprecia en la tabla 7.3, los sesgos y pesos son matrices, vectores y un escalar. Estas variables deben estar almacenadas en memorias ROM de tal forma que al describir en HW totalmente la SRCNN propuesta por Chao Dong et al [47], estas variables no puedan ser modificadas. Como se comentó anteriormente la descripción de HW solo está enfocada en las capas de convolución por ello es que en esta sección se describirá la arquitectura que se propone para almacenar los pesos.

Siendo así pasamos a la segundo punto el cual es la descripción de una memoria ROM que contenga los pesos. Para realizar la descripción de HW, para ello se implementó un programa el cual permite obtener literales del lenguaje Verilog para facilitar la descripción, dicho script se ilustra en la Figura 7.6.

```
function GenMemKerenel(Conv)
  load('modelo.mat')
  if(Conv==1)
     Var=pesos_kernel_conv1;
  end
  if(Conv==2)
    Var=pesos_kernel_conv2;
  end
  if(Conv==3)
   Var=pesos_kernel_conv3;
  end
  [x y z]=size(Var);
  f id=fopen('MemKernel.txt','w');
  aux=0;

  for k=1 : z
    fprintf(fid,'if(page==%d)begin\n',k-1);
    for i=1 : x
      for j=1 : y
        aux=floor(Var(i,j,k)*1000);
        fprintf(fid,'  ');
        fprintf(fid,'Wout%d%d<=%d;   ',i,j,aux);
      end
      fprintf(fid,'\n');
    end
    fprintf(fid,'end\n');
  end

  fclose(fid);
end
```

Figura 7.6, Codigo de la función GenMemKerenel.

La función GenMemKerenel() recibe el valor de 1,2 o 3 de tal forma que genera literales de lenguaje Verilog para realizar una asignación directa de los valorares de los pesos para cada una de las capas y así ayude a realizar la descripción de HW de la memoria ROM necesaria para almacenar cada uno de los pesos. Como últimos detalles de código los valores de los sesgos son tomados directamente de archivo ***modelo.mat*** y las literales obtenidas son almacenadas en un archivo de texto. Finalmente, la arquitectura de la memoria ROM descrita se ilustra en la Figura 7.7.

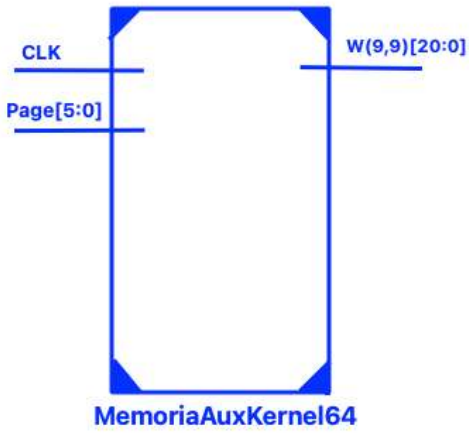


Figura 7.7, Memoria que contiene los Kernels de la primer capa de convolución.

Esta memoria ROM permite almacenar sesenta y cuatro Kernels de 9×9 pixeles, en la siguiente tabla 7.4, se ilustras más información de dicha memoria.

Tabla 7.4, Especificaciones de la memoria MemoriaAuxKernel64.

PINES.	Bus.	Función .	Características.
CLK.	1 bit.	Señal de reloj.	Sensible a flanco de subida.
Page.	6 bits.	Dirección del Kernel.	Dirección valida de 0 a 63.
W(9,9).	81×21 bits.	Obtención del Kernel elegido.	Los datos obtenidos pueden tener signo.

Cabe recalcar que la variable W(9,9), indica que son 9×9 salidas es decir 81 salidas.

7.4 Capa de convolución:

Como ya bien se sabe el algoritmo de la SRCNN que se está tratando en esta tesis cuenta con tres capas de convolución, específicamente utiliza el modelo 9-5-5, donde la primera requiere una convolución que maneje un kernel de 9×9 pixeles, la capa dos y tres requiere una convolución que maneje un kernel de 5×5, por ello es que en esta sección se expondrá la descripción de HW que se propone para tratar estos procesos.

7.4.1 Primer capa de convolución:

En esta sección se expone la arquitectura del procesador de convolución que tratara el proceso de la primer capa de convolución. El procesador que se propone para tratar este proceso es muy similar al tratado en el capítulo cuatro de esta tesis, por ello es qué se siguió el mismo desarrolló de diseño y descripción propuesto en dicho capitulo teniendo en cuenta que ahora el procesador de convolución requiere manipular un kernel de 9×9 pixeles.

Una vez que se desarrolló el diseño y la descripción del HW, se obtuvo una nueva versión de ***Ingrom*** dicha versión recibió el nombre ***Ingrom_*** $\mathbf{9 \times 9}$ y su arquitectura se ilustra en la Figura 7.8.

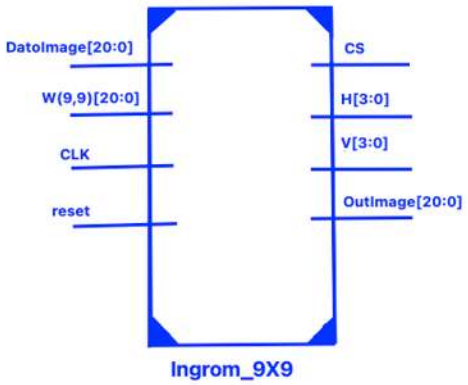


Figura 7.8, Arquitectura de Ingrom_9x9.

Esta versión de ***Ingrom*** opera con un kernel de tamaño de 9×9 pixeles, en la tabla 7.5, se ilustra más información de esta versión.

Tabla 7.5, Especificaciones de Ingrom_9x9.

Pines	Bus	Función	Características
DatoImage	21 bits.	Este bus recibe el pixel que será almacenado en la memoria principal del Procesador.	Los datos recibidos pueden ser con signo.
W(9,9)	81×21 bits.	Estos bues reciben los pixeles del kernel.	Los datos recibidos pueden ser con signo.
CLK	1 bit.	Recibe la señal de reloj.	Fácil de sincronizar.
reset	1 bit.	Reinicia el procesador.	1 = reinicia el procesador. 0 = el procesador trabaja
CS	1 bit.	Permite usarse como señal de reloj o habilitador de una memoria donde se almacenarán los datos obtenidos.	El periodo de esta señal es más lento que la señal de CLK.
H	4 bits.	Este bus permite manejar de manejar una memoria con arquitectura matricial, donde esta almacenada la imagen a manipular.	El rango de este bus es de 1 a 10 de tal forma que solo permite extraer un parche de una imagen.
V	4 bits.	Este bus permite manejar de manejar una memoria con arquitectura matricial, donde esta almacenada la imagen a manipular.	El rango de este bus es de 1 a 10 de tal forma que solo permite extraer un parche de una imagen.
OutImage	21 bits.	Obtiene los datos de la operación de convolución de forma secuencial.	Los datos obtenidos pueden ser con signo.

7.5 Almacenamiento de las imágenes obtenidas

Ahora, que se conoce como tratar el proceso de cada capa de convolución, surge la necesidad de almacenar las imágenes obtenidas en cada capa de convolución, por ello es que en esta sección se expondrá el tipo de arquitectura de memoria que se propone para almacenar las imágenes obtenidas.

7.5.1 Memoria para la primer capa de convolución:

El proceso de la primer capa de convolución obtiene 64 imágenes las cuales son necesarias almacenar para que así pasen a la segunda capa de convolución. La arquitectura de memoria se propone esta dividida en diferentes secciones las cuales se exponen a continuación.

- **Memoria RAM lineal:**

  Principalmente se requiere una memoria RAM que tenga la capacidad de almacenar las 64 imágenes o bien 64 módulos para almacenar cada una de ellas. En este caso se propusieron 64 memorias las cuales tienen la arquitectura que se ilustra en la Figura 7.9.

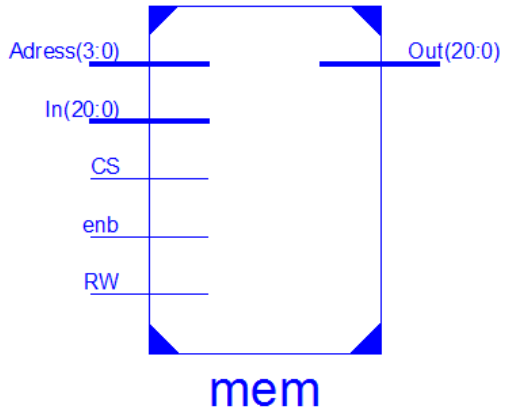


Figura 7.9, Memoria RAM lineal.

Memoria RAM que permite almacenar 100 datos de 21 bits. En la tabla 7.6, se indican más características de esta memoria.

Tabla 7.6, Especificaciones de la memoria RAM lineal.

Pines.	Bus.	Función.	Características.
Adress.	4 bits.	Bus que asigna la dirección de la memoria.	La dirección valida es de 1 a 10.
In.	21 bits.	Bus que recibe el dato que se desea escribir en la memoria.	Puede recibir datos con signo.
CS.	1 bit.	Es la señal de reloj.	Es sensible a flancos de bajada.
enb.	1 bit.	Este pin habilita o deshabilita la memoria.	1=memoria habilitada. 0=memoria deshabilitada.
RW.	1 bit.	Pin para leer o escribir en la memoria.	1=escribe en la memoria. 0=lee la memoria.
Out.	21 bits.	Bus donde se obtienen lo datos al leer la memoria.	Puede obtener datos con signo.

Cabe recalcar que esta memoria es muy pequeña y el tamaño máximo de la imagen que se puede almacenar es una imagen de 10×10 pixeles, sin embargo, esta pensada para poder realizar la memoria y la arquitectura que se propone es la misma solo que pueda almacenar imágenes más grandes.

- **Codificador:**

  Nuevamente se requiere un codificador para que la memoria descrita en la imagen de la Figura 7.9, pueda operar como una memoria con arquitectura matricial, para ello se propone utilizar un codificador con la arquitectura que se ilustra en la Figura 7.10.

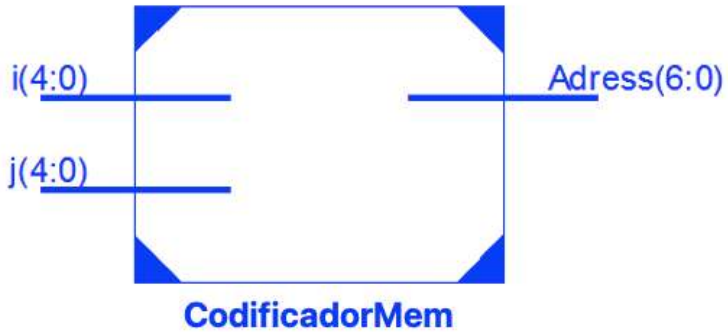


Figura 7.9, Arquitectura del CodificadorMem.

Como se observa el codificador tiene la arquitectura muy similar al de la Figura 7.5, en la tabla 7.7, se ilustra más información de este codificador.

Tabla 7.7, Especificaciones del CodificadorMem.

PINES.	Bus.	Función .	Características.
i	4 bits.	Dirección vertical.	Dato valido 1 a 10.
j	4 bits.	Dirección horizontal.	Dato valido 1 a 10.
Adress.	7 bits.	Se obtiene las direcciones i, j en una sola dirección.	Dato obtenido con un rango de 1 a 100.

- **Decodificador:**

  La arquitectura de memoria que se propone es un poco más compleja que la arquitectura de la memoria ROM de la sección 7.3.1, ya que como se comentó la arquitectura que se propone para almacenar las 64 imágenes obtenidas por el proceso de la primer capa de convolución es necesario indicar cuando se va utilizar cada una de las 64 memorias para ello se propone un codificador con la arquitectura que se ilustra en la Figura 7.8.

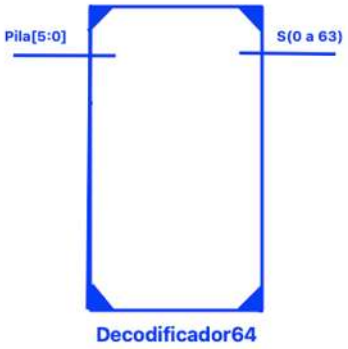


Figura 7.8, Decodificador64.

Este decodificador es el módulo encargado de habilitar cada una de las 64 memorias que contendrá la arquitectura de la memoria que se pretende usar. En la tabla 7.8, se ilustra más información del codificador.

Tabla 7.8, Especificaciones de Decodificador64.

PINES.	Bus.	Función .	Características.
Pila[5:0]	6 bits.	Recibe un dato para decidir que pin se habilitara.	Los datos validos son desde 0 has el 63.
S1 hasta S63	1 bit.	Son los pines que se habilitaran según el dato recibido en la entrada Pila.	Son 64 pines.

- **Multiplexor:**

  Se describió un multiplexor el cual permite obtener solo los datos de una de las 64 memorias, su arquitectura se ilustra en la Figura 7.9.

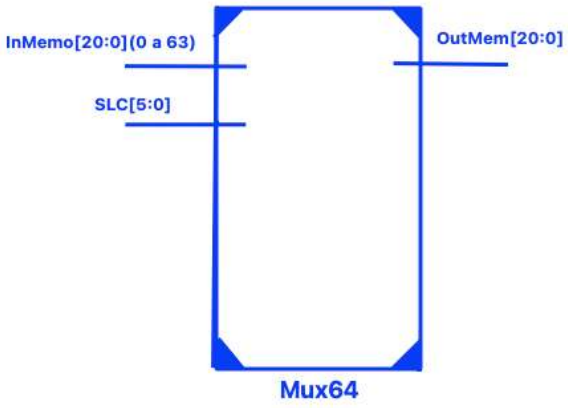


Figura 7.9, Arquitectura de Mux64.

Este multiplexor es el encargado de obtener solo una de las sesenta y cuatro salidas de cada las memorias, como se puede apreciar en la Figura 7.9, en la tabla 7.9, se ilustra más información de este multiplexor.

Tabla 7.9, Especificaciones de Mux64.

PINES.	Bus.	Función.	Características.
InMemo0 hasta InMemo63	21 bits.	Cada uno de estos buses recibe los datos de cada una de las salidas de las 64 memorias.	Son 64 entradas, es decir InMemo0, InMemo1…InMem63.
SLC	6 bits.	Permite elegir qué entrada del multiplexor se obtendrás en la salida.	Dirección valida de 0 a 63.
OutMemo	21 bits.	Es la salida del multiplexor.	

- **Memoria64:**

  Finalmente se unieron todos los módulos descritos anteriormente y se obtuvo una memoria donde se almacenarán las 64 imágenes obtenidas por el proceso de la primera capa de convolución. En la Figura 7.10, se ilustra la arquitectura final que se propone para almacenar dichas imágenes.

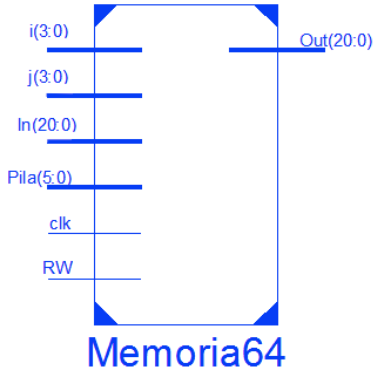


Figura 7.10, Arquitectura de Memoria64.

La memoria que se ilustra en la Figura 7.10, contiene los módulos mencionados anteriormente y esta memoria es la que permite almacenar las 64 imágenes obtenidas por el proceso de la primer capa de convolución. En la tabla 7.10, se muestra más información de esta memoria.

Tabla 7.10, Especificaciones de Memoria64.

PINES.	Bus.	Función.	Características.
i	4 bits.	Dirección vertical.	Dato valido 1 a 10.
j	4 bits.	Dirección horizontal.	Dato valido 1 a 10.
In	21 bits.	Bus que recibe el dato que se desea escribir en la memoria.	Puede recibir datos con signo.
Pila	6 bits.	Este bus decide cual de las memorias internas se va utilizar.	Dirección valida de 0 a 63.
clk	1 bit.	Recibe la señal de reloj.	Es sensible a flancos de bajada.
RW	1 bit.	Pin para leer o escribir en la memoria.	1=escribe en la memoria. 0=lee la memoria.

- **Interfaz para Memoria64**

  Al adquirir la memoria que se ilustra en la Figura 7.10, no se puede conectar directamente al procesador ***Ingrom_9x9*** por ello es que requiere una unidad para que

el procesador pueda manipular directamente esta memoria, la unidad que se utilizó se ilustra en la Figura 7.11 y se le nombro ***ControlMemoria***.

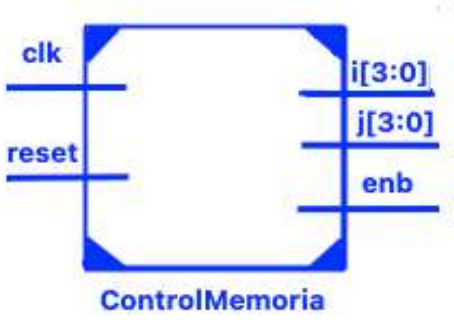


Figura 7.11, ControlMemoria.

La unidad que se ilustra en la Figura 7.11, permite conectar que ***Ingrom*** pueda manipular la memoria ***Memoria64***, en la tabla 7.11 se ilustra más información de esta unida.

Tabla 7.11, Especificaciones de ControlMemoria.

PINES.	Bus.	Función.	Características.
clk	1 bit.	Señal de reloj.	Sensible a flancos de bajada.
reset	1 bit.	Se reinicia la unidad.	1 = reinicia la unidad. 0 = trabaja la unidad.
i	4 bits.	Dirección vertical.	Dato valido 1 a 10.
j	4 bits.	Dirección horizontal.	Dato valido 1 a 10.
enb	1 bit.	Indica cuando tiene la última dirección.	En 1 cuando ***i*** y ***j*** tienen valor de 10.

7.6 Descripción de HW de la primera capa de convolución

Después de haber adquirido todas las unidades que componen la primer capa de convolución se obtuvo el HW compone a esta primera capa, la arquitectura se ilustra en la Figura 7.12.

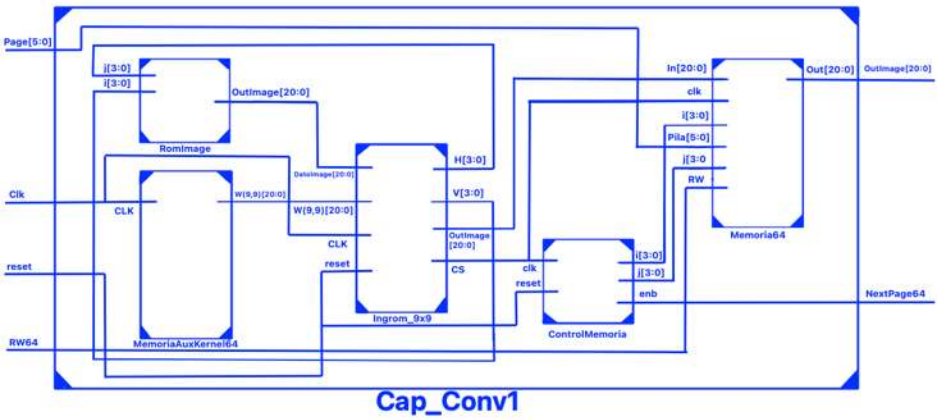


Figura 7.12, Arquitectura de la primer capa de convolución.

El módulo que se ilustra en la Figura 7.12, permite realizar la primer capa de la convolución, en la tabla 7.12, se ilustra más información de este módulo.

7.12, Especificaciones de Cap_Conv1.

PINES.	Bus.	Función.	Características.
clk	1 bit.	Señal de reloj.	Sensible a flacos de subida.
reset	1 bit.	Reinicia la unidad.	1 = reinicia la unidad. 0 = trabaja la unidad.
Page	6 bits.	Entrada para elegir cuál de las 64 memoria se va leer o escribir y que peso se va utilizar.	Datos validos del 0 al 63.
RW	1 bit.	Entrada que decide si se lee o se escribe en ***Memria64***.	1 = Escribe en la memoria. 0 = Lee la memoria.
OutImage	21 bits.	Salida para obtener los datos de ***Memria64***.	Los datos obtenidos pueden tener signo.
NextPage64	1 bit.	Indicador que indica cuando esta lista para pasar a la siguiente pagina.	Solo es una señal.

Cabe recalcar que la salida ***NextPage64*** es solo un indicar el cual está pensado en que le ayude a una unidad de control en tomar la decisión de cuando habilitar o deshabilitar las entradas ***Page***, ***reset*** y ***RW*** cuando se implante todo el algoritmo. También hasta el momento esta descripción de HW solo puede extraer parches con un tamaño de 10×10 pixeles debido que hasta el momento existe el problema de que al realizar la extracción de parches y realizar la operación de convolución existe cierta perdida de información.

Capítulo 8

Resultados

8.1 Introducción

En el presente capítulo, se exponen de los resultados obtenidos en la investigación del diseño y la descripción de las herramientas fundamentales para implementar un algoritmo de SR en una FPGA, donde se narran las reacciones qué se iban obteniendo al adquirir principalmente el HW ***Conv2D, Ingrom***, ***interfaz de imagen, PI y Cap_Conv1***, así como también los resultados del SW que ayudo a describir dicho HW. Para así llegar a obtener el diseño y descripción de HW de estas herramientas fundamentales.

8.2 Resultados del capitulo cuatro

En capitulo cuatro se trato del diseño y descripción de un procesador de convolución, donde se implemento en Matlab una función que realiza la convolución espacial y obtiene literales de lenguaje Verilog que facilita la descripción de HW de la operación de convolución y el procesador de convolución, así como también se obtuvo los recursos que se utilizaron.

8.2.1 Resultados obtenidos de la función *Conv2D*

Observando el código de la Figura 4.6 y recordando lo que ya se explicó de dicho código su funcionamiento se ve de forma abstracta por lo que es mejor es ejecutar el código y explicar lo que hace. La ejecución del código se muestra en la Figura 8.1.

```
>> I=[17 24 1 8 15; 23 5 7 14 16; 4 6 13 20 22; 10 12 19 21 3; 11 18 25 2 9];
>> K=[8 1 6; 3 5 7; 4 9 2];
>> Conv2D(I,K,'F','X','W')
```

Figura 8.1, Ejecución de la función Conv2D.

Como se observa se definen dos matrices, $I[i,j]$ para la imagen y $K[i,j]$ para el kernel, la función recibe a **I** y **K**, se le especifica la variable resultante la cual será **F**, la imagen es

representada por la variable **X** y el kernel por la variable **W**. El resultado de la ejecución se muestra en la Figura 8.2.

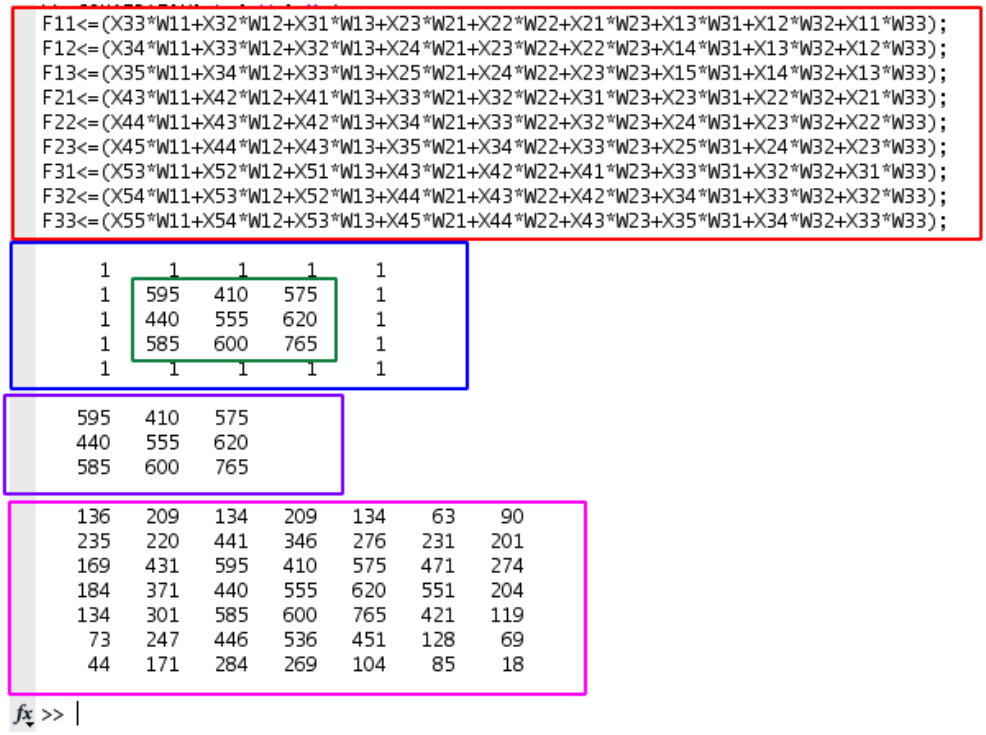


Figura 8.2, resultado de la ejecución de la función Conv2D.

Como se puede ver en la Figura 8.2. El cuadro de color rojo ilustra, la obtención de literales del lenguaje Verilog las cuales son validas para $x(i,j)$ de tamaño 5×5 y $w(i,j)$ de tamaño 3×3 y éstas se describieron directamente en HW, de esta manera nos aseguramos de obtener un sistema más rápido y además se logró implementar la convolución espacial rápidamente. El cuadro de color azul muestra la matriz resultante de la convolución de la imagen y el kernel, dicha matriz es el resultado obtenido a nivel SW de nuestra implementación, como se puede observar que la matriz obtenida en la parte indicada de color rojo tiene un tamaño de 3x3 y vemos que las matriz de color azul es de 5×5 sin embargo Matlab cuenta con una función para la convolución espacial, al ejecutar dicha función el resultado que se obtiene es el de color morado y la de color rosa es la convolución redonda. Si observamos que la convolución resultante de color morado tiene los mismos valores centrales de la matriz obtenida por nuestra implementación es de decir lo que está de color verde y que tiene un

tamaño de 3×3 de la misma forma que la matriz que se implementara en *Verilog*(lo que está dentro del rectángulo rojo).

8.2.2 Resultados de la simulación del módulo Conv2D

Se realizó la simulación con el simulador *Isim* integrado en el *ISE* de *Xilinx*, para la señal de prueba x y el kernel w mostrados a continuación:

$$x = \begin{vmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \\ \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{8} & \mathbf{9} \\ \mathbf{1} & \mathbf{9} & \mathbf{3} & \mathbf{4} & \mathbf{7} \\ \mathbf{8} & \mathbf{7} & \mathbf{4} & \mathbf{3} & \mathbf{2} \\ \mathbf{1} & \mathbf{2} & \mathbf{2} & \mathbf{4} & \mathbf{2} \end{vmatrix}$$

$$w = \begin{vmatrix} \mathbf{8} & \mathbf{1} & \mathbf{6} \\ \mathbf{3} & \mathbf{5} & \mathbf{7} \\ \mathbf{4} & \mathbf{9} & \mathbf{2} \end{vmatrix}$$

La simulación del módulo Conv2D se ilustra en la Figura 8.3.

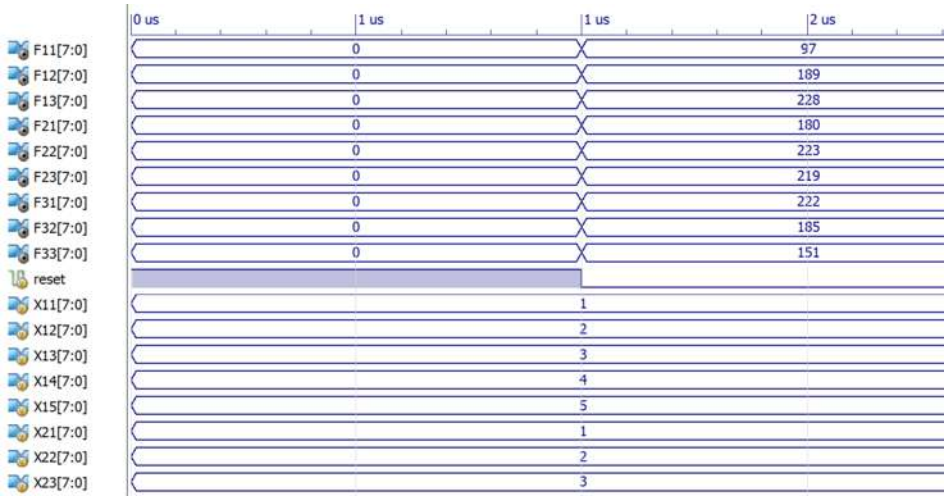


Figura 8.3, simulación del módulo Conv2D.

Como se puede apreciar en la Figura 8.3, el **reset** inicia en un estado lógico de uno por lo que el resultado de la convolución es cero es decir se obtiene una imagen de color negro en un esquema de color RGB o bien la operación de la convolución esta deshabilitada, pero cuándo

está en estado bajo se puede ver que se obtiene una imagen con los valores mostrados a continuación.

$$x * w = \begin{vmatrix} 97 & 189 & 228 \\ 180 & 223 & 219 \\ 222 & 185 & 151 \end{vmatrix}$$

Para comprobar los resultados, se volvió a ejecutar la función *Conv2D* y efectivamente se obtuvo el mismo resultado, el cual se ilustra en la Figura 8.4.

```
>> Conv2D(I,K,'F','X','W')
F11<=(X33*W11+X32*W12+X31*W13+X23*W21+X22*W22+X21*W23+X13*W31+X12*W32+X11*W33);
F12<=(X34*W11+X33*W12+X32*W13+X24*W21+X23*W22+X22*W23+X14*W31+X13*W32+X12*W33);
F13<=(X35*W11+X34*W12+X33*W13+X25*W21+X24*W22+X23*W23+X15*W31+X14*W32+X13*W33);
F21<=(X43*W11+X42*W12+X41*W13+X33*W21+X32*W22+X31*W23+X23*W31+X22*W32+X21*W33);
F22<=(X44*W11+X43*W12+X42*W13+X34*W21+X33*W22+X32*W23+X24*W31+X23*W32+X22*W33);
F23<=(X45*W11+X44*W12+X43*W13+X35*W21+X34*W22+X33*W23+X25*W31+X24*W32+X23*W33);
F31<=(X53*W11+X52*W12+X51*W13+X43*W21+X42*W22+X41*W23+X33*W31+X32*W32+X31*W33);
F32<=(X54*W11+X53*W12+X52*W13+X44*W21+X43*W22+X42*W23+X34*W31+X33*W32+X32*W33);
F33<=(X55*W11+X54*W12+X53*W13+X45*W21+X44*W22+X43*W23+X35*W31+X34*W32+X33*W33);

     1     1     1     1     1
     1    97   189   228     1
     1   180   223   219     1
     1   222   185   151     1
     1     1     1     1     1

    97   189   228
   180   223   219
   222   185   151

     8    17    32    47    62    29    30
    11    28    58   120   154   110    89
    15   101    97   189   228   185   115
    71   113   180   223   219   180    79
    36   123   222   185   151   128    40
    35   111   118    98    83    62    18
     4    17    28    38    48    26     4

fx >>
```

Figura 8.4, ejecución de Conv2D para comprobación de resultado.

De igual manera para comprobar que la operación de convolución descrita en HW, funcione correctamente se realizó el mismo proceso de simulación y comprobación un par de veces más y efectivamente siguió teniendo éxito. Note que como se indicó la salida está limitada a 8 bits las matrices con las que se realizaron las pruebas tienen valores pequeños ya que hasta este momento no se ha restringido a valores de 8 bits es decir que el resultado de la imagen filtrada solamente obtenga valores de 0 a 255.

8.2.3 Recursos utilizados de la operación de convolución

Como ya se aludió anteriormente, la operación de convolución espacial descrita por la ecuación 2.4, tiene un alto costo computacional debido a sus sumas ponderadas; por lo que es de suma importancia conocer los recursos y la cantidad de estos utilizados para llevar a cabo dicha implementación.

El *ISE* de *Xilinx* nos proporciona una estimación de los recursos utilizados al momento que termina la sintetización, la cual para el módulo Conv2D se ilustra en la Figura 8.5.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	3256	5720	56%
Number of fully used LUT-FF pairs	0	3256	0%
Number of bonded IOBs	345	102	338%
Number of DSP48A1s	9	16	56%

Figura 8.5, Estimación de los recursos utilizados.

En la Figura 8.5, se ilustra la estimación para la FPGA *Spartan-6 XC6SLX9*. El sintetizador del *ISE* estima que utiliza el 56% de los recursos de las tablas LUTS, que como se conoce estas son las que realiza las operaciones lógicas en una FPGA, es decir que esta efectúa la operación de convolución descrita; como se sabe el módulo **Conv2D** descrito solo trabaja para una imagen $x(i,j)$ de tamaño 5×5 y un kernel $w(i,j)$ de tamaño 3×3, por lo que si incrementamos principalmente el tamaño de la imagen $x(i,j)$ y opcionalmente el tamaño del kernel $w(i,j)$, requerirá más operaciones lógicas por lo que se exige el uso de más las tablas LUTS, por ejemplo si utilizamos una imagen $x(i,j)$ de tamaño 10×10 y un kernel $w(i,j)$ de tamaño 3×3, los recurso de las tablas LUTS se excederían a más del 100% para la FPGA *Spartan-6 XC6SLX9*, sin embargo existe una gran variedad de FPGAS en el mercado. Es decir, para la familia *Spartan-6*, en la Figura 8.6, se ilustra los recursos disponibles para las diferentes variedades FPGAS de esta familia.

Device	Logic Cells	Total Slices	SLICEMs	SLICELs	SLICEXs	Number of 6-Input LUTs	Maximum Distributed RAM (Kb)	Shift Registers (Kb)	Number of Flip-Flops
XC6SLX4	3,840	600	300	0	300	2,400	75	38	4,800
XC6SLX9	9,152	1,430	360	355	715	5,720	90	45	11,440
XC6SLX16	14,579	2,278	544	595	1,139	9,112	136	68	18,224
XC6SLX25	24,051	3,758	916	963	1,879	15,032	229	115	30,064
XC6SLX45	43,661	6,822	1,602	1,809	3,411	27,288	401	200	54,576
XC6SLX75	74,637	11,662	2,768	3,063	5,831	46,648	692	346	93,296
XC6SLX100	101,261	15,822	3,904	4,007	7,911	63,288	976	488	126,576
XC6SLX150	147,443	23,038	5,420	6,099	11,519	92,152	1,355	678	184,304
XC6SLX25T	24,051	3,758	916	963	1,879	15,032	229	115	30,064
XC6SLX45T	43,661	6,822	1,602	1,809	3,411	27,288	401	200	54,576
XC6SLX75T	74,637	11,662	2,768	3,063	5,831	46,648	692	346	93,296
XC6SLX100T	101,261	15,822	3,904	4,007	7,911	63,288	976	488	126,576
XC6SLX150T	147,443	23,038	5,420	6,099	11,519	92,152	1,355	678	184,304

Figura 8.6, Recursos lógicos de la familia Spartan-6 [48].

Como se ilustra en la Figura 8.6, existen FPGAS que tienen más recurso que otras y como ya se mencionó el módulo de la convolución descrito, principalmente requiere tablas LUTS, por lo que, si se desea que la operación de convolución trabaje con imágenes y kernels del tamaño más grande, solo hay que analizar que FPGA cumpla con los recursos de tablas LUTS suficientes para su implementación.

Ahora bien analizando la Figura 8.6, se puede observar un dato bastante significante para llevar realizar la implementación el cual consiste en las IOBs, que como ya se mencionó en el capítulo 2.5, son los boques de entrada y salida de la FPGA; los cuales son excedidas por un 238%, sin embargo esta estimación no es importante ya que el módulo Conv2D no se pretende implementar directamente y el uso de los bloques IOBs se redujeron en el **diseño de un procesador de convolución** expuesto en el capítulo 4.3.

8.2.4 Estimación de tiempo de la operación de convolución

Como ya se señaló anteriormente nuestro principal interés es que la operación de convolución se ejecute lo más rápido posible. Observemos nuevamente la Figura 8.2, que ilustra dentro del rectángulo de color rojo las literales de lenguaje Verilog que se describieron, teniendo

conocimiento de este lenguaje de descripción HW, se puede comprender que tiene asignaciones no bloqueantes, es decir que las operaciones que conforman la operación de convolución se realizan al instante que la señal de reset trasciende de uno a cero, dicho de otro modo la operación de convolución se realiza cuándo existe un flanco de bajada en la entrada reset. Este contexto se analizar en la Figura 8.3.

En la actualidad la velocidad de los procesadores de propósito general está definida por una fuente de reloj es decir la velocidad de procesador esta determinada con el tiempo que tarda en realizar una operación lógica, dicho tiempo esta definido por la fuente de reloj del procesador. De acuerdo con este contexto se puede realizar una estimación de la velocidad relativa de la operación de convolución, para ello se consideró que la entrada de ***reset*** puede estar sincronizada a una señal que tarde un ciclo de reloj completo en realizar una transición de uno a cero. Dicha señal y la implementación de la operación puede ser implementada en una tarjeta de desarrollo FPGA *Spartan-6 XC6SLX9*, la cual regularmente esta trabaja con una frecuencia de reloj de $50MHz$, es decir que **la velocidad de la ejecución de la operación de convolución en la FPGA** con las características mencionadas anteriormente es de:

$$\boldsymbol{T = \frac{1}{F} = \frac{1}{50MHz} = 20nS.}$$

Ahora bien, analicemos la ejecución de la función conv2 que trae por defecto Matlab. Para ello podemos hacer uso de los comando *tic* y *toc*. En la Figura 8.7, se ilustra la ejecución de los comandos mencionado para analizar la velocidad de ejecución de la función conv2, de igual manera utilizando una imagen $x(i,j)$ de tamaño $5x5$ y un kernel x(i,j) de tamaño $(3x3)$.

```
>> tic; conv2(I,K,'valid'); toc
Elapsed time is 0.000555 seconds.
>>
```

Figura 8.7, Velocidad de ejecución de Conv2.

Como se puede observar en la Figura 8.7, **la velocidad de ejecución es de $\boldsymbol{T = 555\mu S}$**. Dicha velocidad depende de la cantidad de instrucciones que componen la función conv2, además de los recursos disponibles de la computadora donde se ejecuta Matlab. La ejecución qué se ilustrada en la Figura 8.7, se realizó en una computadora con las características que se ilustran en la Figura 8.8.

Figura 8.8, Características del equipo y sistema operativo donde se ejecutó la función convolución de Matlab.

Como se ilustra en la Figura 8.8, la computadora cuenta con un procesador con una frecuencia de reloj de $2.5GHz$, resaltando la frecuencia de reloj disponible en el procesador ya que como se puede apreciar, es muy alta en comparación de la frecuencia considerada anteriormente en la FPGA; pero si existe una tarjeta de desarrollo FPGA con la misma frecuencia de reloj, la velocidad de la ejecución de la operación de convolución descrita será más grande.

Finalmente podemos decir que la velocidad que se ejecuta la operación de convolución descrita en la FPGA es mayor en comparación de la función de convolución en Matlab.

Nota: La comparación no se realizó con la función Conv2D implementada en Matlab, ya que en esta nos interesa adquirir literales de lenguaje *Verilog* y este proceso es aún más lento.

8.2.5 Simulación de Ingrom

Nuevamente teniendo en cuenta la suma importancia de realizar la simulación para determinar si el sistema diseñado funciona correctamente, en esta sección se presenta la simulación del procesador ***Ingrom***. La señal de prueba $x(i,j)$ con la que se realizó la simulación tiene un tamaño de 25×25 y esta almacenada en la memoria **MemImage**, es decir es a la que se le extraerán los parches de tamaño 5×5 , los cuales se trasladan a la memoria principal de ***Ingrom***, y el kernel $w(i,j)$ es introducido por la entradas con la letra w y éste debe tener un tamaño de 3×3. La simulación se muestra en la Figura 8.9.

Nota: la imagen o señal de prueba almacenada en **MemImage** no es del tamaño de las imágenes que se pretende manipula es decir de tamaño 105×105 o más grande, sin embargo, es el tamaño máximo de memoria que el *ISE* de *Xilinx* nos permite diseñar ya que se intento aumentan el tamaño, pero la sintentización tardaba demasiado, pero con una memoria de tamaño de 25×25 y celdas de 8 bits nos basta para realizar la simulación.

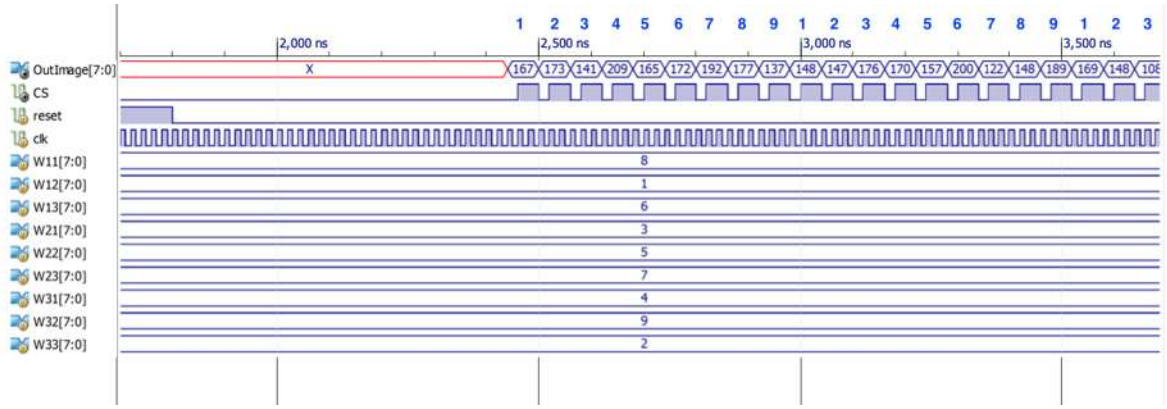
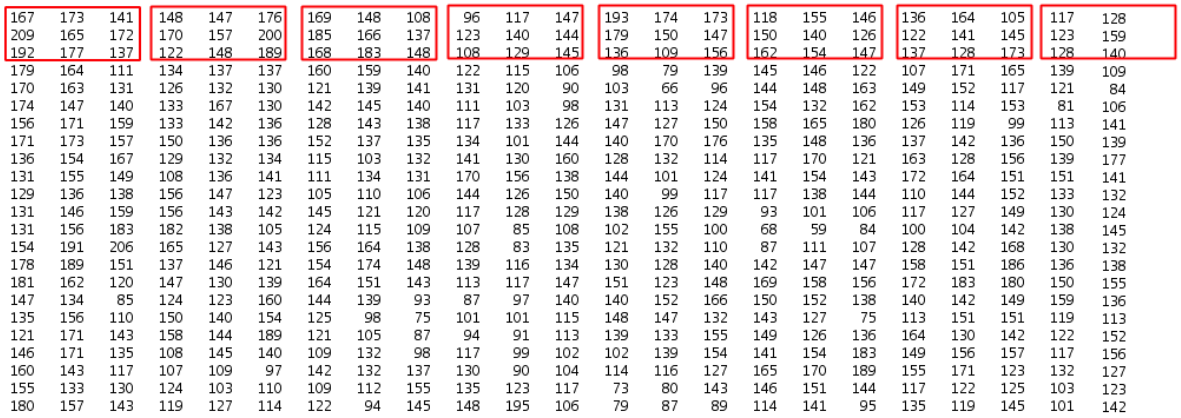


*Figura 8.9, Simulación de **Ingrom**.*

Como se puede observar en la Figura 8.9, se ilustra una parte de la simulación del procesador ***Ingrom*** ya que con esto basta para reportar que ***Ingrom*** funciona correctamente. Para demostrar su funcionamiento, con la ayuda de Matlab se obtuvo la imagen filtrada y el resultado se ilustra en la Figura 8.10.



167	173	141	148	147	176	169	148	108	96	117	147	193	174	173	118	155	146	136	164	105	117	128	
209	165	172	170	157	200	185	166	137	123	140	144	179	150	147	150	140	126	122	141	145	123	159	
192	177	137	122	148	189	168	183	148	108	129	145	136	109	156	162	154	147	137	128	173	128	140	
179	164	111	134	137	137	160	159	140	122	115	106	98	79	139	145	146	122	107	171	165	139	109	
170	163	131	126	132	130	121	139	141	131	120	90	103	66	96	144	148	163	149	152	117	121	84	
174	147	140	133	167	130	142	145	140	111	103	98	131	113	124	154	132	162	153	114	153	81	106	
156	171	159	133	142	136	128	143	138	117	133	126	147	127	150	158	165	180	126	119	99	113	141	
171	173	157	150	136	136	152	137	135	134	101	144	140	170	176	135	148	136	137	142	136	150	139	
136	154	167	129	132	134	115	103	132	141	130	160	128	132	114	117	170	121	163	128	156	139	177	
131	155	149	108	136	141	111	134	131	170	156	138	144	101	124	141	154	143	172	164	151	151	141	
129	136	138	156	147	123	105	110	106	144	126	150	140	99	117	117	138	144	110	144	152	133	132	
131	146	159	156	143	142	145	121	120	117	128	129	138	126	129	93	101	106	117	127	149	130	124	
131	156	183	182	138	105	124	115	109	107	85	108	102	155	100	68	59	84	100	104	142	138	145	
154	191	206	165	127	143	156	164	138	128	83	135	121	132	110	87	111	107	128	142	168	130	132	
178	189	151	137	146	121	154	174	148	139	116	134	130	128	140	142	147	147	158	151	186	136	138	
181	162	120	147	130	139	164	151	143	113	117	147	151	123	148	169	158	156	172	183	180	150	155	
147	134	85	124	123	160	144	139	93	87	97	140	140	152	166	150	152	138	140	142	149	159	136	
135	156	110	150	140	154	125	98	75	101	101	115	148	147	132	143	127	75	113	151	151	119	113	
121	171	143	158	144	189	121	105	87	94	91	113	139	133	155	149	126	136	164	130	142	122	152	
146	171	135	108	145	140	109	132	98	117	99	102	102	139	154	141	154	183	149	156	157	117	156	
160	143	117	107	109	97	142	132	137	130	90	104	114	116	127	165	170	189	155	171	123	132	127	
155	133	130	124	103	110	109	112	155	135	123	117	73	80	143	146	151	144	117	122	125	103	123	
180	157	143	119	127	114	122	94	145	148	195	106	79	87	89	114	141	95	135	119	145	101	142	

Figura 8.10, Imagen filtrada obtenida con la ayuda de Matlab.

Como se puede observar en la Figura 8.10, los primero datos obtenidos ilustrados en el primer rectángulo de color rojo, son iguales a los primero nueve datos obtenidos en la salida de ***OutImage***, que se ilustra en la Figura 8.9, note que de igual forma los datos dentro del segundo rectángulo de color rojo de la Figura 8.10, son iguales a los siguientes nueve datos obtenidos en salida ***OutImage*** de la Figura 8.10, de esta manera se comprobó que para todos los datos obtenidos en la salida ***OutImage*** son iguales a lo obtenido en Matlab, por lo que el procesador ***Ingrom*** funciona de forma eficaz.

En la Figura 8.11, se indica que existe una latencia, desde que la entrada ***reset*** tiene un cero lógico(cuándo opera el procesador) hasta el primer dato de la salida ***OutImage***, ya que en ese tiempo se realiza la primer extracción del parche de la imagen $x(i,j)$, que como ya se habló la extracción se almacena en la memoria principal de ***Ingrom*** y en ese momento es cuándo la memoria principal adquiere los datos de la extracción y se realiza la operación de convolución, las siguientes extracciones se realizan mientras se obtienen los datos de manera serial en ***OutImage*** y dicha latencia ya no aparece, sin embargo internamente la extracción sigue tomando cierto tiempo.

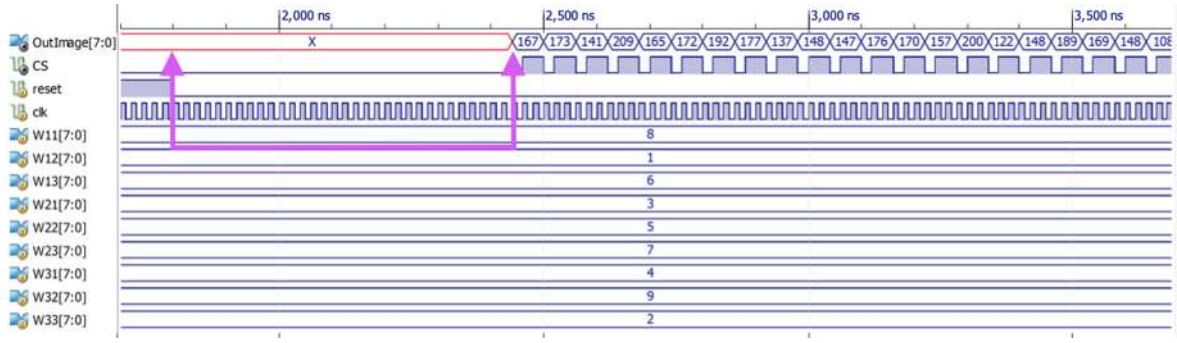


Figura 8.11, ilustración de latencia de la extracción de parches.

Como se puede observar en la Figura 8.11, las flechas de color rosa, indican la latencia a la que se refiere. Observe que la salida ***CS*** cambia de estado una vez que se obtiene los datos en la salida OutImage de esta forma se asegura que la imagen filtrada puede ser almacenada exitosamente.

8.2.6 Recursos utilizados por *Ingrom*

De acuerdo a lo ya mencionado en la sección 8.3 es importante determinar los recursos necesarios para la implementación del diseño, en este caso del diseño de ***Ingrom***. Para ello nuevamente se utilizó la estimación que proporciona el *ISE* de *Xilinx* y se obtuvo lo que se ilustra en la Figura 8.12.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	179	11440	1%
Number of Slice LUTs	2750	5720	48%
Number of fully used LUT-FF pairs	98	2831	3%
Number of bonded IOBs	83	102	81%
Number of BUFG/BUFGCTRLs	2	16	12%
Number of DSP48A1s	9	16	56%

*Figura 8.12, Recursos estimados para la implementación de **Ingrom** en la FPGA.*

En la Figura 8.12, se ilustra la estimación para la FPGA *Spartan-6 XC6SLX9* y como se puede observar el sintetizador estima que se utilizara un 48% de las tablas LUTS y un 81% de entradas y salidas, note que el uso de entradas y salidas se ha reducido notoriamente, esto se logro gracias a la extracción de parches ya que solo ocupa de un bus de entrada de 8 bits para proporcionar los datos a la memoria principal de ***Ingrom***. Observe que los demás recursos lógicos no llegan a la capacidad máxima por lo que hasta este momento si se desea ***Ingrom*** ya puede ser implementado una FPGA *Spartan-6 XC6SLX9*.

8.2.7 Estimación de tiempo de *Ingrom*

Posteriormente se analizó cuanto tiempo tarda en obtener completamente la imagen filtrada, para ello se utilizó el simulador *Isim* propio de *ISE* de *Xilinx*, considerando que ***Ingrom*** se implementara en una tarjeta de desarrollo FPGA con una frecuencia de reloj de $50MHz$, dicha estimación se ilustra en la Figura 8.13.

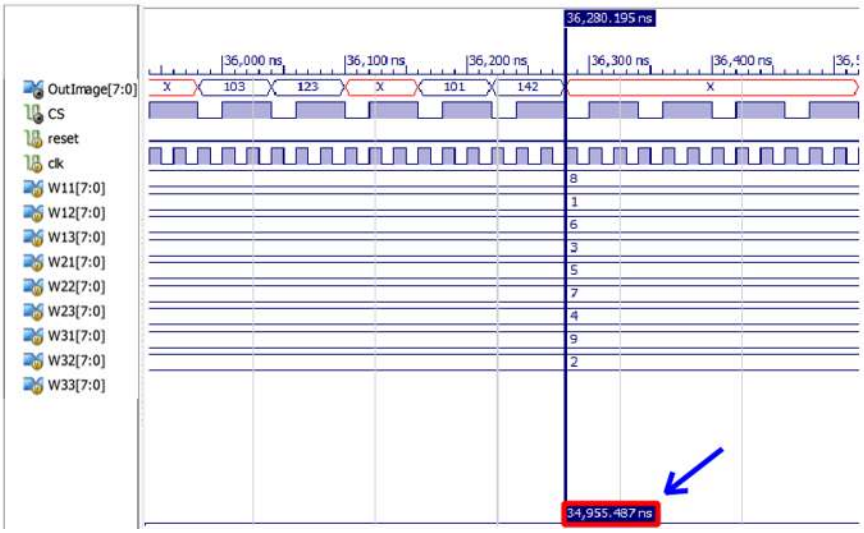


*Figura 8.13, estimación del tiempo la obtención de la imagen filtrada con **Ingrom**.*

En la Figura 8.13, se ilustra la estimación del tiempo de la obtención de la imagen filtrada tiempo desde que se le aplica un cero lógico al ***reset*** y la obtención del último dato de la imagen filtrada, dicho tiempo se estimó mediante la ayuda de los cursores que tiene el simulador *Isim*. El tiempo se ilustra dentro del rectángulo de color rojo e indicado con una flecha de color azul. Como se observa, el tiempo que tarda en obtener la **imagen filtrada es de 34,955.487ns.**

Nuevamente se analizó la ejecución de la función ***Conv2*** que trae por defecto Matlab, para determinar la velocidad que tarda en obtener una imagen filtrada y así compararla con la velocidad que tarda en obtener la imagen filtrada el procesador ***Ingrom***. La ejecución que se ilustra en la Figura 8.14, se realizó con una señal de prueba o imagen $x(i,j)$ de tamaño $25 \times$

25 y un kernel $w(i,j)$ de tamaño 3×3 ya que con estas mismas características el procesador ***Ingrom*** obtuvo la imagen filtrada.

```
>> tic; conv2(I,K,'valid'); toc
Elapsed time is 0.013045 seconds.
```

Figura 8.14, velocidad de la obtención de la imagen filtrada con la función Conv2.

Como se ilustra en la Figura 8.14, **la velocidad de Conv2 es de 13.045mS, en cambio la velocidad de Ingrom es de 34.955µS**, por lo que nuevamente se puede decir que el procesador **Ingrom** obtiene la imagen filtrada más rápido que la función Conv2.

8.3 Resultados del capítulo cinco

En el capitulo cinco se trató del diseño y descripción de HW de una interfaz de imagen, donde se describió el HW que sea capaz de leer una imagen con el formato ***BMP***, por ello es que en esta sección se muestran los resultados obtenidos para dicha interfaz.

8.3.1 Simulación del módulo ReadRGB

Es necesario saber si la interfaz de imagen cumple con el objetivo de leer una imagen con el formato ***BMP***, por ello es que en esta sección se presenta la simulación del módulo ReadRGB, para determinar si funciona correctamente.

Para realizar la simulación se utilizó una imagen con el formato ***BMP*** y de tamaño de $5x5$ pixeles la cual se ilustra en la Figura 8.15.

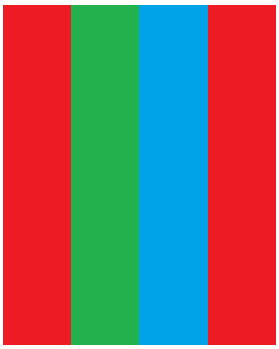


Figura 8.15, Imagen de tamaño $5x5$.

Con la ayuda del **SW HxD Hex Editor** se extrajeron los datos en hexadecimal de la imagen ilustrada en Figura 8.15, dichos datos se ilustran en la Figura 8.16.

```
42 4d 86 00 00 00 00 00 00 00 36 00 00 00 28 00
00 00 05 00 00 00 05 00 00 00 01 00 18 00 00 00
00 00 50 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 24 1c ed 4c  b1 22 e8 a2 00 24
1c ed  ff  ff  ff  00  24 1c ed 4c b1 22 e8 a2 00 24
1c ed  ff  ff  ff  00  24 1c ed 4c b1 22 e8 a2 00 24
1c ed  ff  ff  ff  00  24 1c ed 4c b1 22 e8 a2 00 24
1c ed  ff  ff  ff  00  24 1c ed 4c b1 22 e8 a2 00 24
1c ed  ff  ff  ff  00
```

Figura 8.16, Datos en hexadecimal de la imagen BMP de tamaño $5x5$.

Los datos ilustrados en la Figura 8.16, se almacenaron el archivo de textos con formato **dat**, el cual se sabe es leído con el comando **$readmeh** y que permite almacenarlos directamente en la memoria descrita en la Figura 5.2.

Una vez establecido los datos de la imagen a manipular se realizó la simulación en el Isim de Xilin, la simulación se muestra en al Figura 8.17.

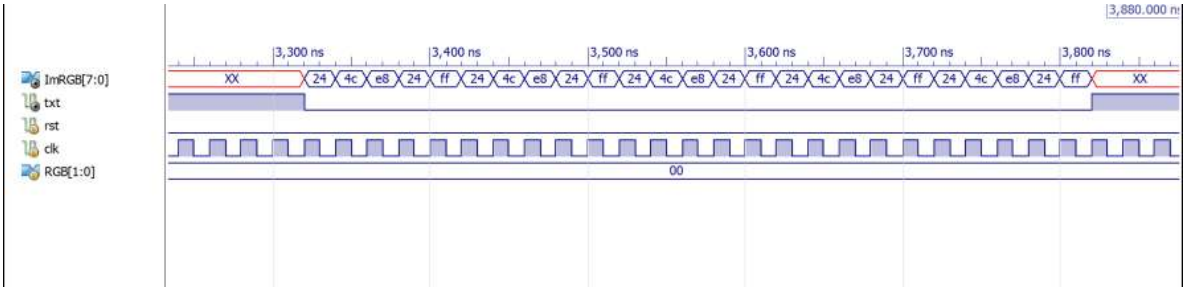


Figura 8.17, simulación del módulo ReadRGB.

Para demostrar que el módulo funciona correctamente se muestra la extracción del canal **R**, observe que la entrada ***RGB*** ilustrada en la Figura 8.17, contiene el dato 00_2, el cual como ya se indicó en la tabla 5.7, es la instrucción para extraer el canal ***R***, observe que la salida **txt** en un intervalo de tiempo tiene el estado en cero dicho intervalo es cuándo se lee por primera vez la memoria que almacena la extracción del canal, por lo que ese intervalo de tiempo es el que nos interesa por el momento, los datos mostrados en la salida ***ImRGB*** en ese intervalo

de tiempo son los datos del canal rojo por lo que los podemos comparar con los datos que se ilustra en la Figura 8.18.

1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	
24	1c	ed	4c	b1	22	e8	a2	00	24	1c	ed	ff	ff	ff	00
24	1c	ed	4c	b1	22	e8	a2	00	24	1c	ed	ff	ff	ff	00
24	1c	ed	4c	b1	22	e8	a2	00	24	1c	ed	ff	ff	ff	00
24	1c	ed	4c	b1	22	e8	a2	00	24	1c	ed	ff	ff	ff	00
24	1c	ed	4c	b1	22	e8	a2	00	24	1c	ed	ff	ff	ff	00

Figura 8.18, Datos que corresponden a la imagen.

La Figura 8.18, muestra los datos que corresponde como tal a la imagen, los cuales fueron extraídos con la ayuda de la tabla 5.1, del formado ***BMP***, ilustrado en la Figura 8.15(recordando que ultimo renglón, corresponde al primer renglón de la imagen), como se puede observar se le agregado un renglón ilustrado de color rojo, verde y azul el cual contiene los valores 1, 2 y 3, con la finalidad de ilustrar que cada columna corresponde a un canal del esquema ***RGB***, note que aparece un dato extra con el valor de cero ese valor corresponde a un salto de línea el cual indica que los datos siguientes corresponde a otro renglón de la imagen. Como podemos ver los datos obtenidos en la salida ***ImRGB***, de la Figura 8.17, corresponden al canal **R**, el cual es el que se deseaba extraer. Este mismo proceso se realizó para comprobar que se estaban extrayendo los datos del canal **G** y **B**, también se realizó el mismo proceso, para diferentes imágenes y la extracción del canal deseado se extrajo exitosamente, por lo que el módulo **ReadRGB** funciona correctamente.

8.3.2 Recursos utilizados por la interfaz de imagen

Los recursos que nos interesan son solo los del módulo ***Contromem*** ilustrado en la Figura 5.9, ya que este es el núcleo de la implementación de la interfaz imagen y como ya se habló la descripción de los demás módulos que componen el módulo ***ReadRGB*** ilustrado en la Figura 5.9 y 5.10, se describieron con la finalidad de realizar la simulación. Los recursos utilizados se ilustran en la Figura 8.19.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	188	11440	1%
Number of Slice LUTs	497	5720	8%
Number of fully used LUT-FF pairs	184	501	36%
Number of bonded IOBs	30	102	29%
Number of BUFG/BUFGCTRLs	1	16	6%

Figura 8.9, recursos utilizados de Contromem.

Como se puede ver en la Figura 8.19, el bloque ***Contromem*** sí se puede implementar una FPGA *Spartan-6 XC6SLX9*.

8.3. 2 Estimación de tiempo de *ReadRGB*

Finalmente se analizó cuanto tiempo tarda en extraer un canal de la imagen de la Figura 8.15, para ello se utilizó el simulador *Isim* propio de *ISE* de *Xilinx*, considerando que el módulo **Contromem** se implementara en una tarjeta de desarrollo FPGA con una frecuencia de reloj de $50MHz$, la estimación se ilustra en la Figura 8.10.

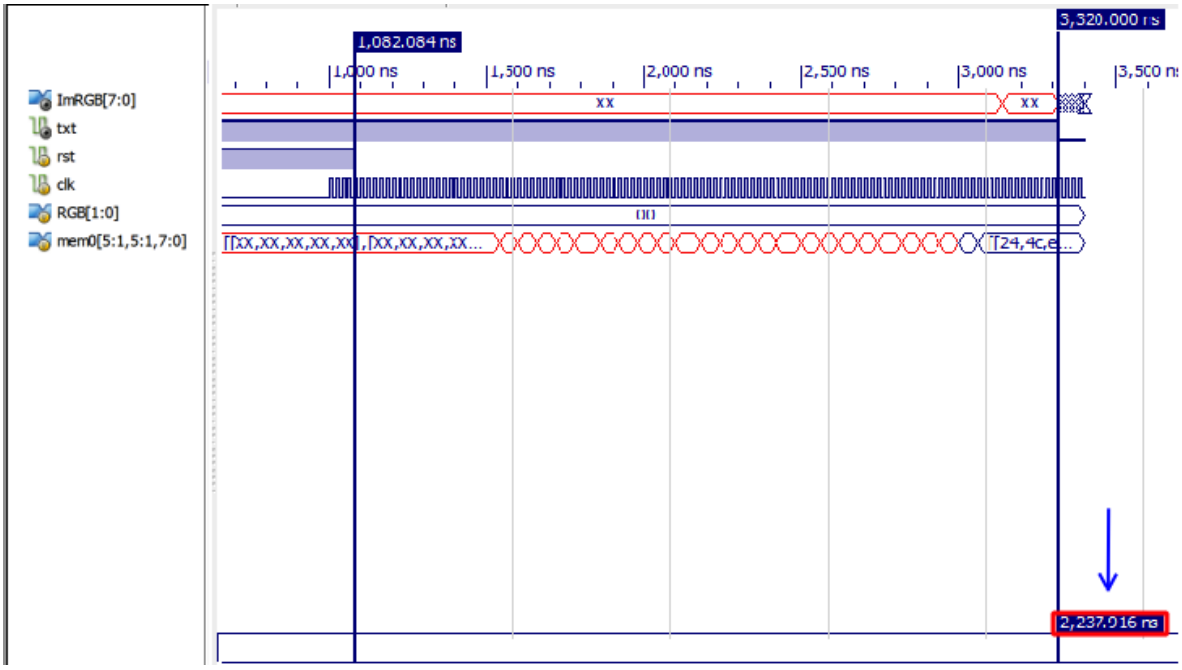


Figura 8.10, Estimación del tiempo la obtención de cana de la imagen de tamaño 5x5.

En la Figura 5.16, se observa el tiempo de estimación de la extracción del canal de la imagen, el cual se estimo mediante el uso de los cursores y esta ilustrado, dentro del rectángulos de

color rojo e indicado con una flecha color azul, el tiempo es de **2,237.016nS**, observe que esta estimación considera el tiempo desde que la entrada en ***rst*** se coloca en cero dado que como ya se indicó qué es cuándo opera el sistema y hasta cuándo termina almacenar los datos de la extracción en la memoria donde se almacena dicha extracción.
Esta estimación fue comparada con la extracción de un canal de la misma imagen en Matlab, dicha estimación se realizó como se ilustra en la Figura 8.11.

```
>> tic; I = imread('imagen5x5pixeles.bmp'); R=I(:, :, 3); toc;
Elapsed time is 0.010157 seconds.
```

Figura 8.11, Tiempo de ejecución de la extracción en de un canal de imagen de tamaño 5*x*5.

Como se puede observar en al Figura 8.18, se obtiene el tiempo de la extracción en Matlab el cual es de **10.157ms**, recordando que el tiempo de la extracción del canal en descripción de HW, es de **2.2370µS**, por lo podemos decir que la extracción del canal es mas rápida que la extracción en Matlab.

8.4 Resultados del capitulo seis

En capitulo seis se trato del diseño y descripción de HW de un ISP, donde se describió el HW que es capaz procesar una imagen, por ello es que en esta sección se presenta el procesamiento de una imagen para demostrar el funcionamiento de PI.

Para demostrar el funcionamiento del ISP, se realizó mediante la simulación del ISP ***PI,*** para procesar una imagen y se comparo con el procesamiento de la misma imagen realizado en la plataforma *Matlab*. En la simulación, se utilizó el simulador Isim propio del *ISE* de *Xilinx* y el procesamiento de Matlab se realizó mediante un script. En las siguientes secciones, se presentan los pasaos que se llevaron para realizar tanto la simulación, la comparación y los recursos utilizados por ***PI***.

8.4.1 Declaración de las variables con la que se realizaron las pruebas

Las pruebas realizadas presentadas en este capítulo se realizaron con una imagen de tamaño $25x25$ pixeles, con la finalidad de que los canales extraídos puedan ser almacenados por el módulo Memoria. La imagen se ilustra en la Figura 8.12.

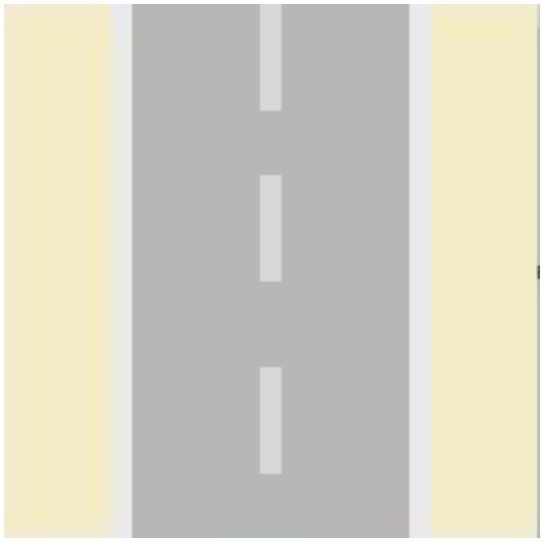


Figura 8.12, Imagen de tamaño $25x25$ *pixeles.*

Las pruebas realizadas y descritas en la siguiente sección tuvieron el enfoque de procesar una imagen. El procesamiento consiste en detectar los bordes de imagen de la Figura 8.12.
El kernel que permite realizar la detestación de bordes de la imagen de tamaño $25x25$ pixeles, se ilustran en la Figura 8.12.

$$w = \begin{vmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{vmatrix}$$

Figura 8.13, Kernels de interés [49],

8.4.2 Declaración de las entradas del ISP PI

La declaración de las variables de entrada del ISP PI se realizaron directamente en el simulador Ism, dicha declaración se ilustra en la Figura 8.14.

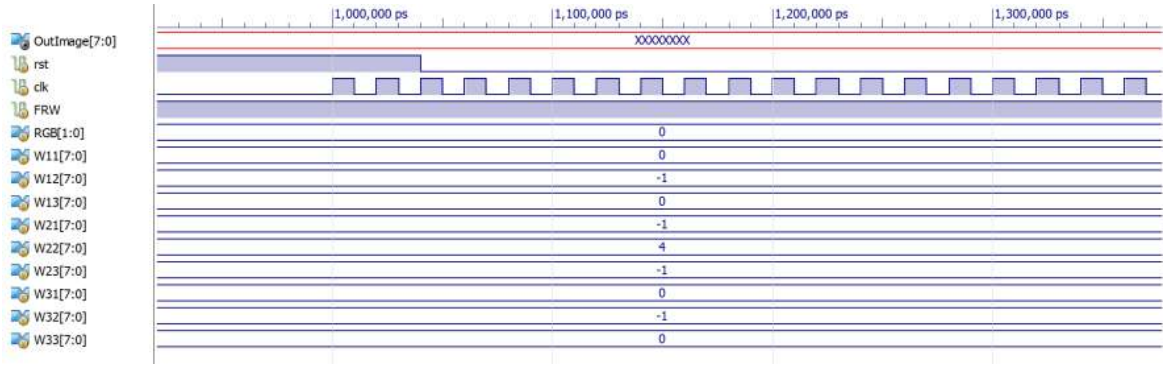


*Figura 8.14, Declaración de las entradas del ISP **PI**.*

En la Figura 6.10, se observa la declaración de las entradas del ISP ***PI***, como se puede ver las entradas indicadas con la letra **W**, representan las entradas del kernel, las cuales coinciden con los valores del kernel de la Figura 8.13. La entrada ***rst*** se sabe que es la entrada de *reset* del ISP, la cual inicialmente está en uno lógico, como se puede ver en un determinado tiempo se pone en cero lógico y se mantiene hasta que se desea. La entrada **FRW** se conoce que es una entrada que permite leer y escribir en la memoria **MemImageFilter**, observe que esta inicialmente tiene un uno lógico la cual permite escribir en la memoria **MemImageFilter**, cabe mencionar que no es de nuestro interés leer ya que el *Isim* permite visualizar los datos en forma vectorial, por lo que en la salida **OutImage** no abra ningún dato de salida. posteriormente con la ayuda del programa HXD se extrajeron los datos de la imagen en formato hexadecimal y se almacenaron en el archivo con formato **dat**. Observe que la entrada **RGB** inicialmente tiene un dato en cero y como sabe este dato cambia según el canal a extraer, en el siguiente paso se menciona cada cunado se definió esta entrada.

8.4.3 Ejecución de la simulación

La simulación radicó de tres partes, primeramente, se realizó el procesamiento del canal **R**, después el procesamiento canal **G** y finalmente se realizó el procesamiento del canal **B**. Cabe recalcar que la entrada **RGB** se le introdujo el dato correspondiente al canal procesar. Los resultados obtenidos de la simulación se ilustran en las figuras 8.15. 8.16 y 8.17.

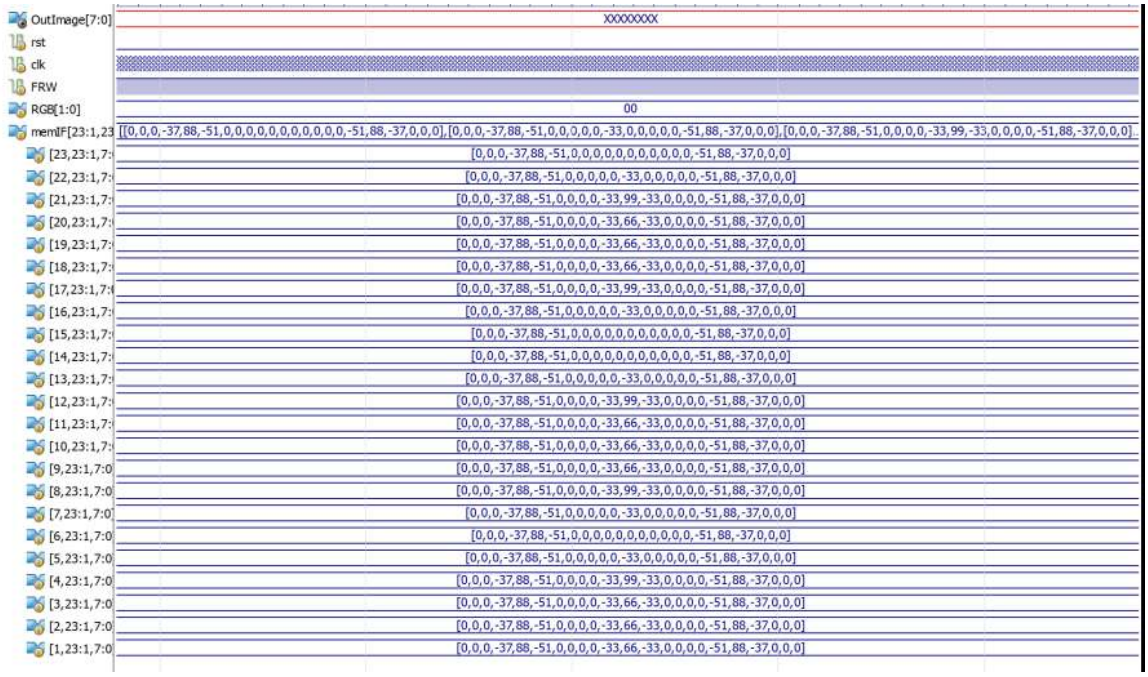


*Figura 8.15, Simulación del ISP **PI** para la detección de bordes del canal R.*

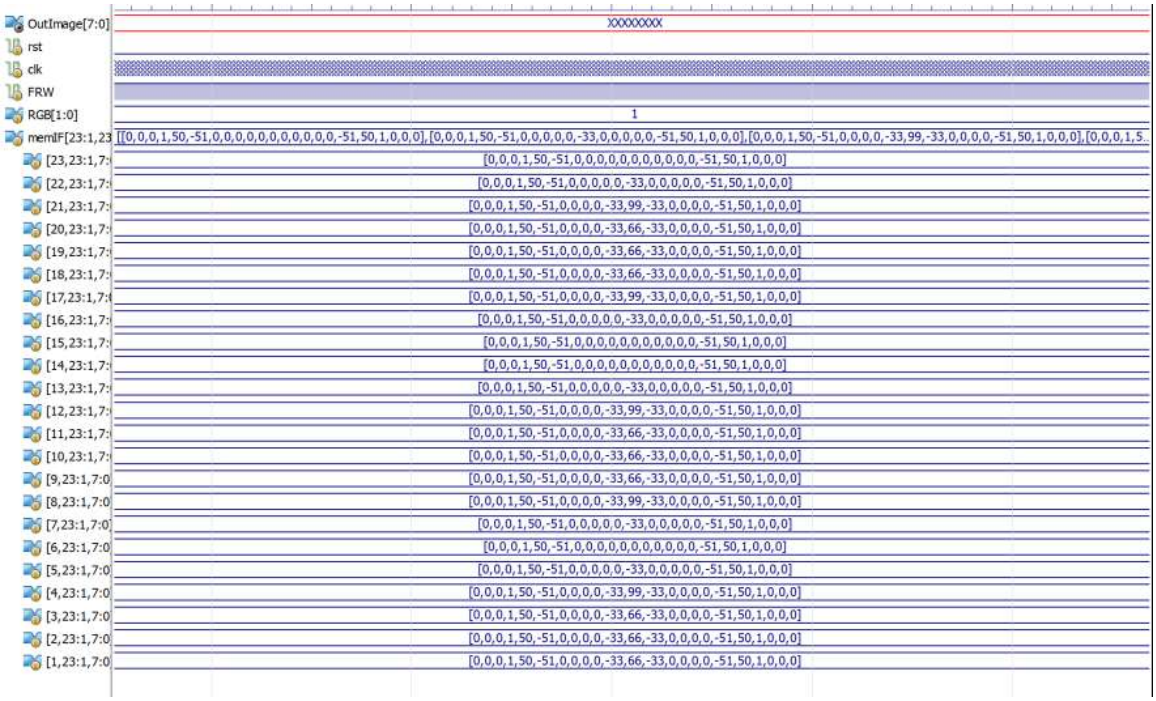


*Figura 8.16, Simulación del ISP **PI** para la detección de bordes del canal G.*

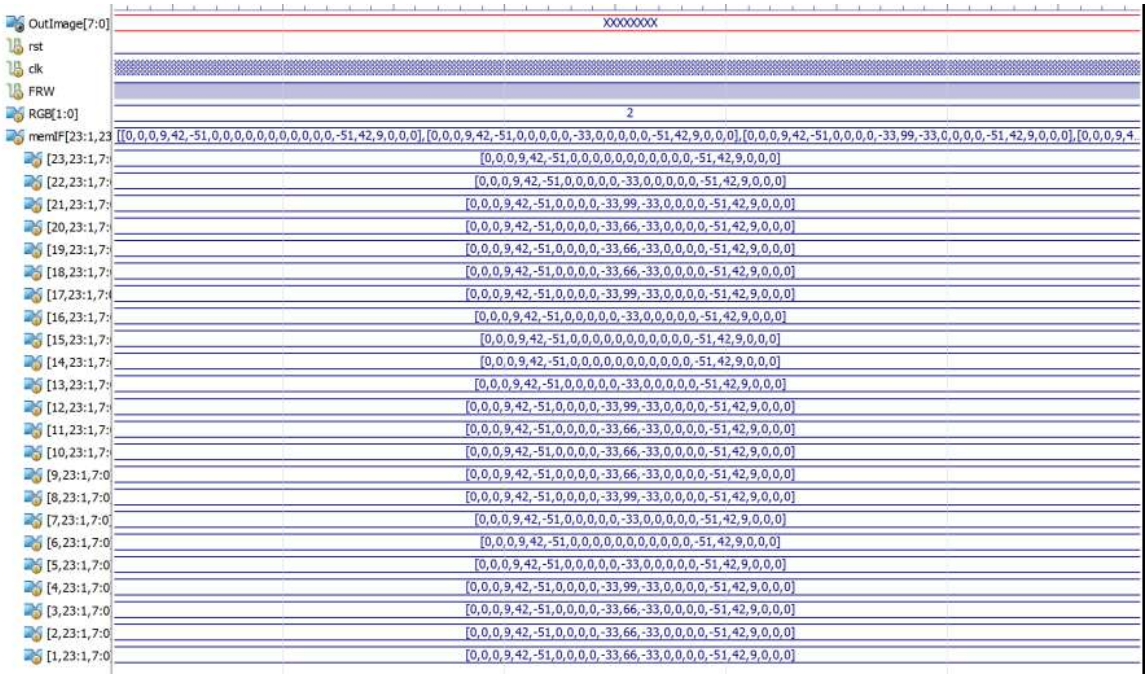


*Figura 8.17, Simulación del ISP **PI** para la detección de bordes del canal B.*

En la Figura 8.15, se ilustra el resultado de la simulación del procesamiento del canal **R** de la Imagen de tamaño $25x25$, en cambio en la Figura 8.16, se ilustra el procesamiento del canal **G** y en la Figura 8.17, se muestra el procesamiento del canal **B**. Cabe recalcar que los datos de la matriz resultante de canal se leen inicialmente con el vector uno que esta en la parte superior y se leen de derecha a izquierda.

8.4.4 Comparación de la simulación ISP *PI* con el procesamiento en Matlab

Una vez que se obtuvo el procesamiento de cada canal de la imagen de tamaño $25x25$ en el paso anterior, se compararon con el procesamiento en Matlab mediante el script que se muestra en la Figura 8.18.

```
clear;
clc;
W=[0 -1 0; -1 4 -1; 0 -1 0];
I  =  imread('PruebaCarretera.bmp');
R=conv2(I(:,:,3),W,'valid')
G=conv2(I(:,:,2),W,'valid')
B=conv2(I(:,:,1),W,'valid')
```

Figura 8.18, Script para realizar el procesamiento de lo canales R, G y B en Matlab.

En la Figura 8.18, se ilustra el Script con el que se realizó el procesamiento de los canales **R**, **G** y **B** en *Matlab*. Como se puede observar se definió el kernel en la variable **W**, posteriormente se lee la imagen con el comando imread y se almacena en la variable I, finalmente se realizó el procesamiento de cada canal de la mediante la función *conv2*. Se ejecutó el script tal como se ilustra en las Figura 8.18, y se obtuvieron los datos de la Figura 8.19, 8.20 y 8.21.

0	0	0	-37	88	-51	0	0	0	0	-33	66	-33	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	-33	66	-33	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	-33	66	-33	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	-33	99	-33	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	0	-33	0	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	0	0	0	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	0	-33	0	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	-33	99	-33	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	-33	66	-33	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	-33	66	-33	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	-33	66	-33	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	-33	99	-33	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	0	-33	0	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	0	0	0	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	0	0	0	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	0	-33	0	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	-33	99	-33	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	-33	66	-33	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	-33	66	-33	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	-33	66	-33	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	-33	99	-33	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	0	-33	0	0	0	0	0	-51	88	-37	0	0	0
0	0	0	-37	88	-51	0	0	0	0	0	0	0	0	0	0	0	-51	88	-37	0	0	0

Figura 8.19, Procesamiento en Matlab para la detección de bordes del canal R.

En la Figura 8.19, se ilustra el procesamiento de Matlab de canal **R** y se comparó con el resultado de la Figura 8.15, y se determinó que efectivamente los valores coinciden en ambos casos.

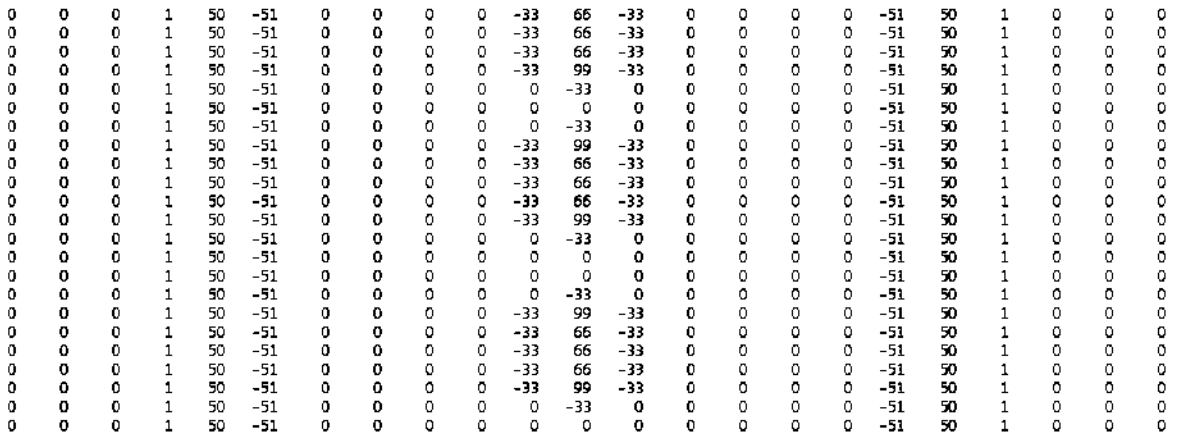


Figura 8.20, Procesamiento en Matlab para la detección de bordes del canal G.

En la Figura 8.20, se ilustra el procesamiento de Matlab de canal **G** y de igual forma se comparó con el resultado de la Figura 8.16, y se determinó que efectivamente los valores coinciden en ambos casos.

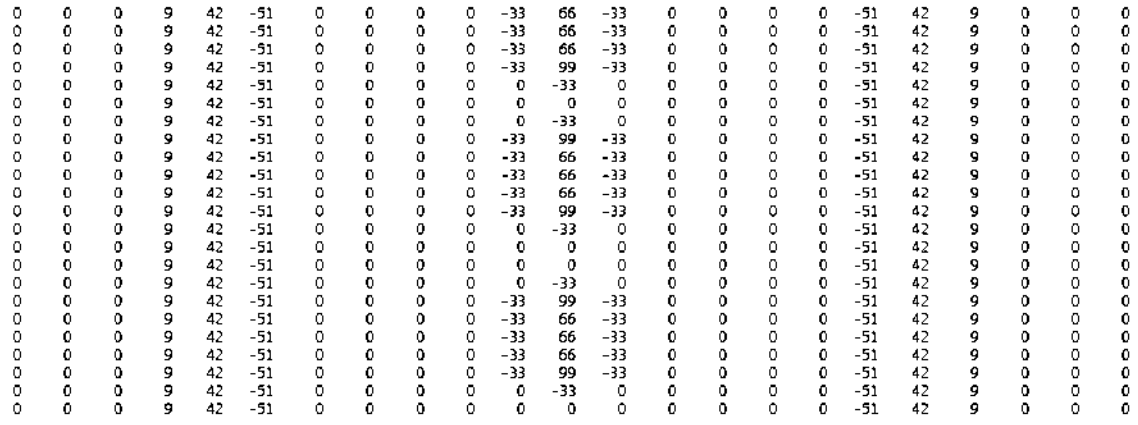


Figura 8.21, Procesamiento en Matlab para la detección de bordes del canal B.

En la Figura 8.21, se ilustra el procesamiento de Matlab de canal **G** y de igual forma se comparo con el resultado de la Figura 8.17, y se determino que efectivamente los valor coinciden en ambos casos.

8.4.5 Obtención de la visualización de la imagen procesada tanto del ISP *PI* y como en Matlab

Para visualizar la imagen obtenida de la simulación del ISP ***PI***, se realizó mediante Matlab ya que en el *ISE* de *Xilinx* no existe una forma de unir los tres canales obtenidos e ilustrarlos en forma gráfica.

El proceso que se realizó consta de los siguientes pasos:

- **Extracción de los datos manualmente**

  En este paso se extrajeron manualmente los datos de los canales obtenidos en la simulación del ISP ***PI***, los cuales se ilustran en las figuras 8.15, 8.16 y 8.17, para ser introducidos en Matlab. Dichos datos de almacenaron en las variables **R**, **G** y **B**. Es decir, se extrajeron los datos de la simulación de cada canal y se almacenaron en las variables correspondientes a cada canal.

- **Conversión de los datos a unit8**

  Teniendo el conocimiento que las imágenes tienen un esquema de color de 8 bits, es decir valores que estén dentro de 0 y 255, pero como podemos observar los datos obtenidos en la simulación del PSI ***PI*** existen valores negativos, los cuales se encuentran fuera del rango ya mencionando y la función que se utilizó en Matlab requiere que los datos estén dentro de este rango, por ello es que previamente se realizó una conversión con la función *cast*, por ejemplo para convertir la variable de canal R se realizó de la siguiente forma: **cast(R,'uint8').**

- **Concatenación de los canales RGB**

  La función con la que se realizó la ilustración de la Imagen procesada en Matlab, necesita que los canales se encontraren almacenados en un objeto que contenga los tres canales, por ello es que antes de utilizar dicha función se realizó una concatenación de los tres canales, este proceso se realizó de la siguiente forma: RGB=cast(3,B,G,R);

- **Ilustración de la imagen procesada con el ISP PI.**

  Se utilizó la función *ishow* para mostrar la imagen almacenada en la variable **RGB**, la cual se conoce que es la variable que contiene la imagen procesada con ISP ***PI***, este proceso se realizó de la siguiente forma imshow(RGB) y se obtuvo la imagen que se ilustra en la Figura 8.22.

*Figura 8.22, Imagen obtenida de la simulación del ISP **PI**.*

En la Figura 8.22, se observa la imagen procesada con la simulación del ISP ***PI***, observe la que el propósito se cumplió ya que los bordes de la Figura 8.12, se resaltan con colores claros en cambio las superficies lisas se atenúan a un color negro.

- **Comprobación de la ilustración de la imagen procesada por ISP *PI* con *Matlab***

  Finalmente se comprobó que la imagen obtenida de la simulación del ISP ***PI***, fuera la misma qué con el procesamiento en Matlab de la Imagen de la Figura 8.12. El procesamiento de Matlab completo, se realizó agregándole al script ilustrado en la Figura 8.18, las líneas de que código que se ilustran la Figura 8.23.

```
ImagenProcesada=cat(3,B,G,R);
imshow(RGB)
```

Figura 8.23, Complemento del script que permite obter la deteción de bordes de la imagen de tamaño $25x25$.

La Figura 8.23, se ilustran las líneas de código que se agregaron al script de la Figura 8.18, el cual permitió obtener el procesamiento de la imagen de tamaño de $25x25$ de forma nativa en *Matlab*. En la Figura 8.24, se ilustra la imagen procesada directamente en *Matlab*.

Figura 8.24, Imagen obtenida con el procesamiento realizado en Matlab.

En la Figura 8.24, se ilustra la imagen obtenida con el procesamiento realizado en Matlab, la cual como se observa es idéntica a la de la Figura 8.22, así que definitivamente se determinó que la detección de bordes realizó satisfactoriamente.

8.4.6 Recursos utilizados del SPI *PI*

Se realizó la estimación de los recursos utilizados para determinar si el SPI ***PI***, puede ser implementados en la FPGA *Spartan-6 XC6SLX9*, parar ello nuevamente se utilizó la estimación de sintetizador, la cual se ilustra en la Figura 8.25.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	5538	11440	48%
Number of Slice LUTs	6765	5720	118%
Number of fully used LUT-FF pairs	1521	10782	14%
Number of bonded IOBs	85	160	53%
Number of BUFG/BUFGCTRLs	8	16	50%
Number of DSP48A1s	9	16	56%

*Figura 8.25, Estimación de los recursos lógicos utilizados por el ISP **PI**.*

En la Figura 8.25, se ilustran los recursos utilizados por ISP ***PI***, como se puede observar todas las secciones están por debajo del 100% con excepción, del uso de las tablas LUTS, estas son excedidas con un 18%, por lo que el diseño del ISP ***PI***, no puede ser implementado en la, FPGA *Spartan-6 XC6SLX9*, sin embargo, puede ser implementada en algún otra FPGA como es la *Spartan-6 XC6SLX16*, la cual como se puede ver en la Figura 8.6, esta cuenta 159.30% más de recursos de tablas LUTS.

Por otra parte, se analizaron los recursos utilizados por cada módulo que compone el ISP ***PI***, para determinar si se podía optimizar y así poder implementarlo en la FPGA *Spartan-6 XC6SLX16*. Las estimaciones que nos interesan son el uso de los recursos de las tablas LUTS, las cuales son ilustradas en la tabla 8.1.

Tabla 8.1, Porcentaje del uso de las tabalas LUTS.

Módulo	Porcentaje del uso de tablas LUTS
ReadRGB	13%
Memoria	40%
Mux	0%
Ingrom	57%
ControlMemoria	0%
MemImageFilter	35%

8.4.7 Estimación del tiempo del procesamiento de un canal de una imagen

Para estimar el tiempo de procesamiento se utilizó el simulador *Isim* propio de *ISE* de *Xilinx*, considerando que el ISP ***PI*** se implementara en una tarjeta de desarrollo FPGA con una frecuencia de reloj de $50MHz$, la estimación obtenida se ilustra en la Figura 8.26.

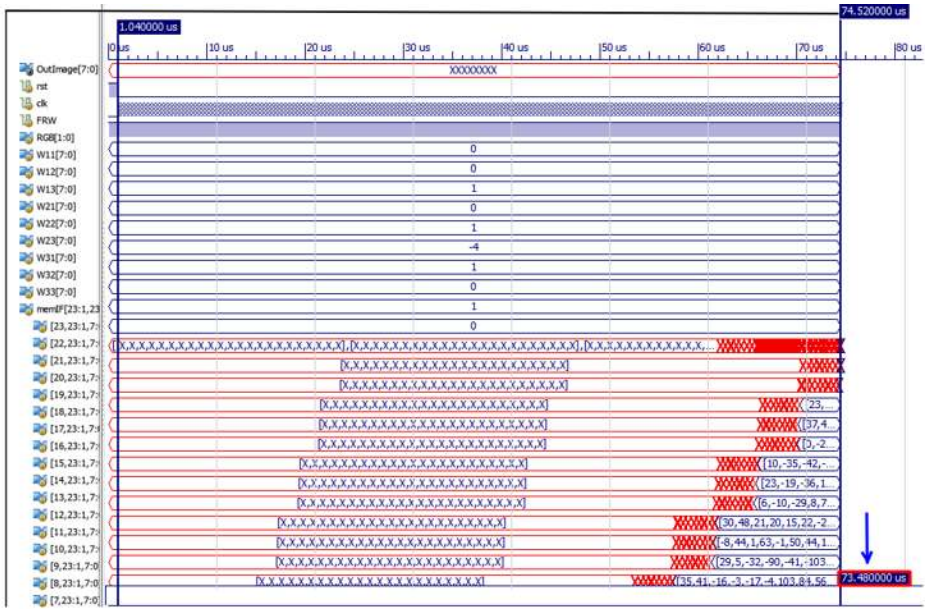


Figura 8.26, Estimación del tiempo para la obtención del procesamiento de un canal de una imagen tamaño $25x25$.

En la Figura 8.26, se observa el tiempo de estimación del procesamiento de un canal de una imagen de tamaño $25x25$. Como se puede observar el tiempo se estimo mediante el uso de los cursores y esta ilustrado, dentro del rectángulos de color rojo e indicado con una flecha color azul, el tiempo es de **73.480000uS**, observe que esta estimación se considera desde el momento que la entrada en **rst** se coloca en cero(es cuándo opera el ISP ***PI***) y hasta cuándo los datos del procesamiento se terminan de almacenar en la memoria.

La estimación fue comparada con el procesamiento en ***Matlab***, de un canal de una imagen con el tamaño $25x25$. Dicha estimación se realizó como se ilustra en la Figura 8.27.

```
>> tic; I = imread('rositas25x25.bmp'); R=I(:, :, 3); conv2(R,W,'valid'); toc;
Elapsed time is 0.016001 seconds.
>>
```

Figura 8.27, Estimación en Matlab del tiempo de procesamiento de un canal de una imagen de tamaño $25x25$.

Como se puede observar en al Figura 8.27, se obtiene el tiempo del procesamiento de un canal en Matlab el cual es de **16.001ms**, en cambio el tiempo del procesamiento del canal con el ISP ***PI***, es de **73.48µS**, por lo que se determina que el procesamiento con el ISP ***PI*** es mas rápido que con el que se realizó en Matlab.

8.5 Resultados del capitulo siete

En capitulo siete se trato del diseño y descripción de HW de la primera capa de convolución del algoritmo SRCCN que se presento en el capitulo tres, como bien se ha visto en las secciones anteriores es de suma importancia conocer el funcionamiento del HW descrito por ello es que en esta sección se presenta el funcionamiento del SOC ***Cap_Conv1***. Para conocer si funciona la descripción de este HW, se realizo en tres fases las cuales se presentan en las siguientes secciones.

8.5.1 Simulación de *Cap_Conv1*

En esta sección se presenta la primer fase para saber si funciona correctamente la descripción del HW ***Cap_Conv1*** la cual se realizo mediante el uso del SW Modelsim debido a que el ISE de Xilinx ya no sintetizaba el modulo ***Memoria64*** ilustrada en la Figura 7.10 y el para sintetizar el procesador ***Ingrom_9x9*** tardaba demasiado tiempo, por ello es que se opto por usar el SW Modelsim.

La simulación consintió en realizar dos procesos de ***Cap_Conv1*** la cual es realizar dos operaciones de convolución con una imagen de prueba, que se muestra en la Figura 8.27 y dos kernels; que como ya se sabe se encuentran almacenados en la memoria ***MemoriaAuxKernel64*** ilustrada en la Figura 7.12.

Figura 9.27, Imagen de prueba.

La imagen que se ilustra en la Figura 9.27, se encuentra almacenada con el formato **dat** en la memoria ***RomImage*** que se encuentra en la arquitectura de ***Cap_Conv1*** la cual se ilustra en

la Figura 7.12. Esta imagen es el canal **Y** en un esquema de color YCbCr el cual como se sabe es con el esquema de color con el que trabaja el algorimo de interes.

Una vez que se definio la imagene a la que se le aplicara las dos operaciones de convolucion mediante ***Cap_Conv1*** se definieron las entradas para este modulo, la definicion de las entradas para el modulo que se ilustra en la Figura 7.12, se ilustra en las tablas 8.2, y 8.3.

Cabe recalcar que como ya se sabe **Cap_Conv1** hasta el momento no puede realizar la operación de convolución con la imagen completa y solo puede realizarla para un parche de la imagen de tamaño de $10x10$ pixeles la cual es una sección de la imagen de la Figura 9.27, es decir una sección como se muestra en la Figura 9.28.

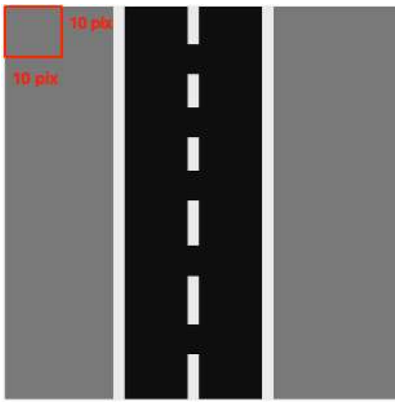


Figura 8.28, Parche elegido.

Como se puede ver en la Figura 8.28, se muestra un cuadro de color rojo el cual es solo el parche con el que trabaja ***Cap_Conv1*.**

Tabla 8.2, Asignación de pines de Cap_Conv1 para la primer operación de convolución.

PINES.	Asignación	Función.
clk	Fuente de reloj.	Señal de reloj.
reset	0_2	Trabaja la unidad.
Page	0_2	Se elige la primera memoria de las 64 y se utiliza el primer peso.
RW	1_2	Se escribe en la primer memoria de las 64 memorias.
OutImage	Salida	Salida para obtener los datos de ***Memria64***.
NextPage64	Salida.	Indicador que indica cuando esta lista para pasar a la siguiente pagina.

Tabla 8.3, Asignación de pines de Cap_Conv1 para la segunda operación de convolución.

PINES.	Asignación	Función.
clk	Fuente de reloj.	Señal de reloj.
reset	0_2	Trabaja la unidad.
Page	1_2	Se elige la segunda memoria de las 64 y se utiliza el primer peso.
RW	1_2	Se escribe en la segunda memoria de las 64 memorias.
OutImage	Salida	Salida para obtener los datos de ***Memria64***.
NextPage64	Salida.	Indicador que indica cuando esta lista para pasar a la siguiente pagina.

Las tablas 8,2 y 8.3, muestra las asignaciones de los pines para ***Cap_Con1*** para realizar dos operaciones de convicción y cada asignación es un caso. Por ello es que se realizo una simulación para cada uno de los casos y los resultados se ilustran en la figuras 8.29 y 8.30.

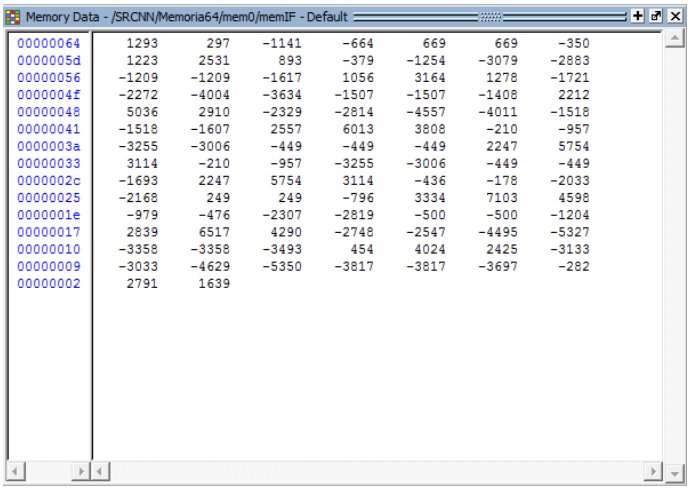


Figura 8.29, Resultados de la primera operación de convolución.

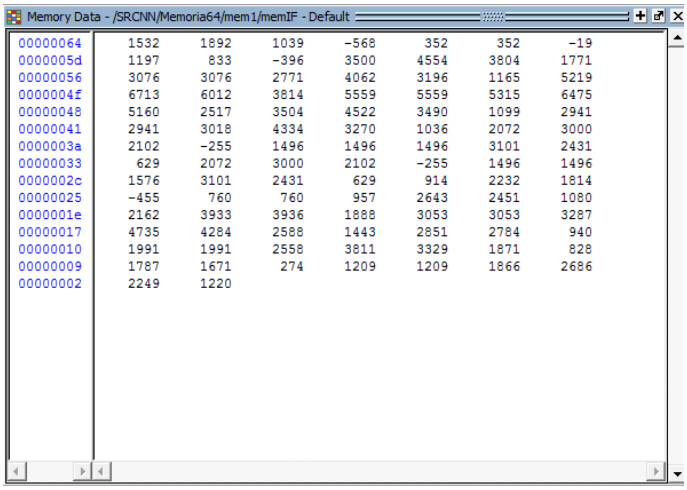


Figura 8.30, Resultados de la segunda operación de convolución.

En la Figura 8.29 y 8.30, se muestran los resultados para cada uno de los dos casos mencionados, como se puede ver estos resultados están representados por una matriz los cuales son lo datos que contienen las dos primeras memorias de ***Memoria64*** y los datos de estas memorias fueron fácil de visualizar gracias a que Modelsim pose con una herramienta de poder manipulad de forma grafica memorias que se encuentran dentro de un modulo HDL.

8.5.2 Resultados de *Cap_Conv1* mediante Matlab

Como se ha visto en la sección anterior hemos obtenido el resultado de ***Cap_Conv1*** sin embargo es necesario saber si los resultados obtenidos son correctos por ellos es que se realizó un scrip en Matlab que ayudo a realizar las mismas dos operaciones de convolución teniendo en cuenta que se utilizó la misma imagen y los mismos kernels. El srrip se ilustra la Figura 8.32.

```
function Cap_Conv1(Conv)
  load('ImagenIB.mat')
  load('modelo.mat')
  R10X10=R([1:10],[1:10]);
  W=pesos_kernel_conv1(:,:,Conv);
  RE=conv2(R10X10,W,'same')
end
```

Figura 8.31, Scrip Cap_Conv1.

En la Figura 8.31, se muestra el scrip que se utilizó, como se puede ver es una función el cual recibe un valor entre uno y sesenta y cuatro, como se puede ver dentro de la función la línea "load('ImagenIB.mat')" es la encargada de cargar la imagen que se ilustra en la Figura 8.27, la línea "load('modelo.mat')" carga el modelo de entrenamiento SRCNN es decir se cargan los pesos y sesgos, en la línea "R10X10=R([1:10],[1:10]);" se realiza el proceso que se ilustra en la Figura 8.29, la línea " W=pesos_kernel_conv1(:,:,Conv);" elige uno de los sesenta y cuatro kernels, según sea el valor que se le da a la función y finalmente con la línea "RE=conv2(R10X10,W,'same')" se realiza la operación de convolución. El resultado obtenido para cado uno de los casos se ilustra en las figuras 8.32 y 8.33.

```
>> Cap_Conv1(1)
RE =

  10×10 single matrix

   0.1647   0.2800  -0.0273  -0.3687  -0.3805  -0.3805  -0.5340  -0.4620  -0.3026  -0.3126
   0.2434   0.4035   0.0466  -0.3481  -0.3343  -0.3343  -0.5315  -0.4485  -0.2537  -0.2740
   0.4300   0.6529   0.2853  -0.1189  -0.0482  -0.0482  -0.2804  -0.2294  -0.0465  -0.0970
   0.4610   0.7117   0.3350  -0.0779   0.0270   0.0270  -0.2151  -0.2018  -0.0165  -0.0425
   0.3127   0.5770   0.2264  -0.1673  -0.0427  -0.0427  -0.2987  -0.3237  -0.0942  -0.0198
   0.3127   0.5770   0.2264  -0.1673  -0.0427  -0.0427  -0.2987  -0.3237  -0.0942  -0.0198
   0.3820   0.6026   0.2572  -0.1589  -0.1499  -0.1499  -0.3994  -0.4541  -0.2801  -0.2318
   0.2920   0.5048   0.2225  -0.1393  -0.1490  -0.1490  -0.3619  -0.3990  -0.2260  -0.1711
   0.1286   0.3175   0.1068  -0.1604  -0.1194  -0.1194  -0.2870  -0.3067  -0.1243  -0.0371
   0.0901   0.2540   0.1233  -0.0340   0.0682   0.0682  -0.0653  -0.1131   0.0306   0.1300
```

Figura 8.32, Resultado de la primer operación de convolución mediante la función Cap_Conv1.

```
>> Cap_Conv1(2)

RE =

  10×10 single matrix

   0.1228   0.2258   0.2696   0.1877   0.1221   0.1221   0.0284   0.1680   0.1795   0.0834
   0.1880   0.3339   0.3822   0.2571   0.2005   0.2005   0.0953   0.2795   0.2860   0.1450
   0.2598   0.4296   0.4749   0.3302   0.3070   0.3070   0.1903   0.3949   0.3944   0.2171
   0.1092   0.2464   0.2658   0.0973   0.0779   0.0779  -0.0440   0.1829   0.2245   0.0924
   0.0642   0.2447   0.3118   0.1595   0.1518   0.1518  -0.0237   0.2118   0.3014   0.2083
   0.0642   0.2447   0.3118   0.1595   0.1518   0.1518  -0.0237   0.2118   0.3014   0.2083
   0.1048   0.3284   0.4349   0.3035   0.2961   0.2961   0.1116   0.3505   0.4535   0.3514
   0.2528   0.5172   0.6489   0.5331   0.5576   0.5576   0.3828   0.6025   0.6724   0.5227
   0.1174   0.3206   0.4073   0.2784   0.3091   0.3091   0.1784   0.3815   0.4564   0.3508
  -0.0389   0.0842   0.1206  -0.0009   0.0365   0.0365  -0.0558   0.1049   0.1901   0.1539
```

Figura 8.33, Resultado de la segunda operación de convolución mediante la función Cap_Conv1.

8.5.3 Comparación de los resultados de *Cap_Conv1*

Como ya se mencionó en la sección anterior es necesario saber si el HW descrito del ***Cap_Conv1*** obtuvo los resultados correctos por ello es que en esta sección se expone como se realizó la comparación de los resultados obtenidos en las dos secciones anteriores.

Si comparamos los resultados obtenidos de las figuras 8.29 y 8.32 así como también los de las figuras 8.30 y 8.33, los resultados no son ni muy cercanos, pero recordemos de que las figuras 8.29 y 8.30, tienen una codificación de punto fijo por lo que los datos obtenidos por el HW ***Cap_Conv1*** hay que descodificarlos, dicha decodificación es muy sencilla ya que hay que dividir todos los datos ente diez mil. Para ello se utilizó Matlab y los resultados descodificados se ilustran en las figuras 8.34 y 8.35.

0.1639	0.2791	-0.0282	-0.3697	-0.3817	-0.3817	-0.5350	-0.4629	-0.3033	-0.3133
0.2425	0.4024	0.0454	-0.3493	-0.3358	-0.3358	-0.5327	-0.4495	-0.2547	-0.2748
0.4290	0.6517	0.2839	-0.1204	-0.0500	-0.0500	-0.2819	-0.2307	-0.0476	-0.0979
0.4598	0.7103	0.3334	-0.0796	0.0249	0.0249	-0.2168	-0.2033	-0.0178	-0.0436
0.3114	0.5754	0.2247	-0.1693	-0.0449	-0.0449	-0.3006	-0.3255	-0.0957	-0.0210
0.3114	0.5754	0.2247	-0.0449	-0.0449	-0.0449	-0.3006	-0.3255	-0.0957	-0.0210
0.3808	0.6013	0.2557	-0.1607	-0.1518	-0.1518	-0.4011	-0.4557	-0.2814	-0.2329
0.2910	0.5036	0.2212	-0.1408	-0.1507	-0.1507	-0.3634	-0.4004	-0.2272	-0.1721
0.1278	0.3164	0.1056	-0.1617	-0.1209	-0.1209	-0.2883	-0.3079	-0.1254	-0.0379
0.0893	0.2531	0.1223	-0.0350	0.0669	0.0669	-0.0664	-0.1141	0.0297	0.1293

Figura 8.34, Resultados decodificados de la primera operación de convolución.

0.1220	0.2249	0.2686	0.1866	0.1209	0.1209	0.0274	0.1671	0.1787	0.0828
0.1871	0.3329	0.3811	0.2558	0.1991	0.1991	0.0940	0.2784	0.2851	0.1443
0.2588	0.4284	0.4735	0.3287	0.3053	0.3053	0.1888	0.3936	0.3933	0.2162
0.1080	0.2451	0.2643	0.0957	0.0760	0.0760	-0.0455	0.1814	0.2232	0.0914
0.0629	0.2431	0.3101	0.1576	0.1496	0.1496	-0.0255	0.2102	0.3000	0.2072
0.0629	0.2431	0.3101	0.1496	0.1496	0.1496	-0.0255	0.2102	0.3000	0.2072
0.1036	0.3270	0.4334	0.3018	0.2941	0.2941	0.1099	0.3490	0.4522	0.3504
0.2517	0.5160	0.6475	0.5315	0.5559	0.5559	0.3814	0.6012	0.6713	0.5219
0.1165	0.3196	0.4062	0.2771	0.3076	0.3076	0.1771	0.3804	0.4554	0.3500
-0.0396	0.0833	0.1197	-0.0019	0.0352	0.0352	-0.0568	0.1039	0.1892	0.1532

Figura 8.35, Resultados decodificados de la segunda operación de convolución.

Ahora bien, comparando visualmente los resultados obtenidos para cada caso o bien la comparación de resultados ilustrados con las figuras 8.34 y 8.32, así como también los resultados de las figuras 8.35 y 8.33, podemos ver que los resultados son muy cercanos para cada caso, pero para ver que tanto se aplicó la ecuación 8.1, la cual describe el error que hay entre dos matrices. Dicho error entres más cercano sea a cero las matrices son casi iguales. Los resultados se ilustran en las figuras 8.36 y 8.37.

$$E = |M_1 - M_2| \qquad (8.1)$$

0.0008	0.0009	0.0009	0.0010	0.0012	0.0012	0.0010	0.0009	0.0007	0.0007
0.0009	0.0011	0.0012	0.0012	0.0015	0.0015	0.0012	0.0010	0.0010	0.0008
0.0010	0.0012	0.0014	0.0015	0.0018	0.0018	0.0015	0.0013	0.0011	0.0009
0.0012	0.0014	0.0016	0.0017	0.0021	0.0021	0.0017	0.0015	0.0013	0.0011
0.0013	0.0016	0.0017	0.0020	0.0022	0.0022	0.0019	0.0018	0.0015	0.0012
0.0013	0.0016	0.0017	0.1224	0.0022	0.0022	0.0019	0.0018	0.0015	0.0012
0.0012	0.0013	0.0015	0.0018	0.0019	0.0019	0.0017	0.0016	0.0013	0.0011
0.0010	0.0012	0.0013	0.0015	0.0017	0.0017	0.0015	0.0014	0.0012	0.0010
0.0008	0.0011	0.0012	0.0013	0.0015	0.0015	0.0013	0.0012	0.0011	0.0008
0.0008	0.0009	0.0010	0.0010	0.0013	0.0013	0.0011	0.0010	0.0009	0.0007

Figura 8.36, Error de la primera operación de convolución.

0.0008	0.0009	0.0010	0.0011	0.0012	0.0012	0.0010	0.0009	0.0008	0.0006
0.0009	0.0010	0.0011	0.0013	0.0014	0.0014	0.0013	0.0011	0.0009	0.0007
0.0010	0.0012	0.0014	0.0015	0.0017	0.0017	0.0015	0.0013	0.0011	0.0009
0.0012	0.0013	0.0015	0.0016	0.0019	0.0019	0.0015	0.0015	0.0013	0.0010
0.0013	0.0016	0.0017	0.0019	0.0022	0.0022	0.0018	0.0016	0.0014	0.0011
0.0013	0.0016	0.0017	0.0099	0.0022	0.0022	0.0018	0.0016	0.0014	0.0011
0.0012	0.0014	0.0015	0.0017	0.0020	0.0020	0.0017	0.0015	0.0013	0.0010
0.0011	0.0012	0.0014	0.0016	0.0017	0.0017	0.0014	0.0013	0.0011	0.0008
0.0009	0.0010	0.0011	0.0013	0.0015	0.0015	0.0013	0.0011	0.0010	0.0008
0.0007	0.0009	0.0009	0.0010	0.0013	0.0013	0.0010	0.0010	0.0009	0.0007

Figura 8.37, Error de la segunda operación de convolución.

En la Figura 8.36, se muestra el resultado obtenido al comparar los resultados obtenidos para la primera operación de convolución, es decir se le aplico la ecuación 8.1, para las matrices que se ilustra en las figuras 8.34 y 8.32, por lo que se puede el error es muy cercano a cero por lo que se puede decir que se obtuvo el mismo resultado, cabe recalcar que el resultado que se obtuvo mediante Matlab es decir el de la 8.32 es mejor ya que Matlab usa Más decimales. De igual manera se le aplico la misma ecuación a las matrices que se ilustran en

las figuras 8.35 y 8.33, por lo que se obtuvo el error que se ilustra en la Figura 8.37 y como se puede ver el error es muy cercano por lo que prácticamente los resultados son iguales.

En esta sección se realizaron dos operaciones de convolución para demostrar como es que opera ***Cap_Conv1,*** pero cabe recalcar que este puede realizar sesenta y cuatro operaciones distintas según se le asigne uno de los sesenta y cuatro pesos.

8.5.4 Estimación del tiempo de la operación de convolución con Cap_*Conv1*

Para estimar el tiempo de una operación de convolución para el **SOC *Cap_Conv1*** se utilizó el simulador ModelSim, considerando que ***Cap_Conv1*** utiliza una frecuencia de reloj de $50MHz$, la estimación obtenida se ilustra en la Figura 8.38.

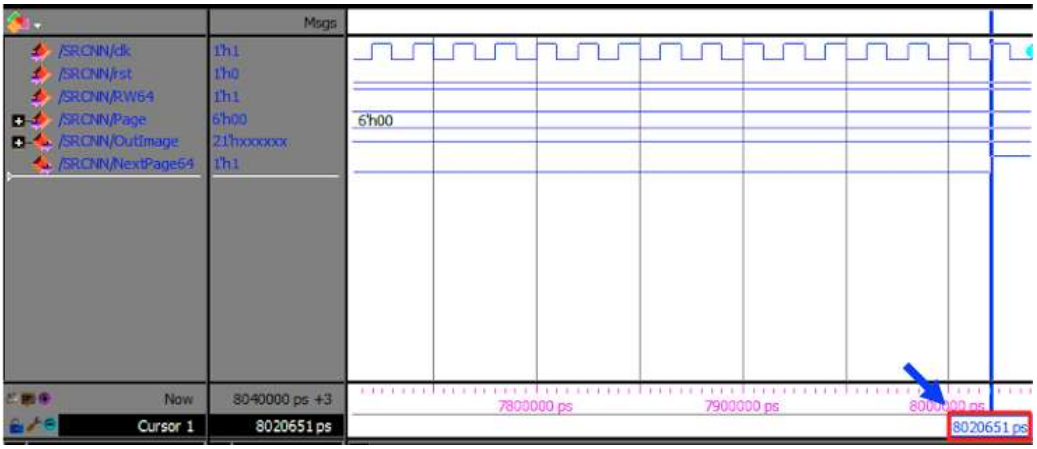


Figura 8.38, Estimación de tiempo de una operación de convolución con Cap_Conv1.

En la Figura 8.38, se observa el tiempo de estimación de una operación de convolución con ***Cap_Conv1***, el cual se estimó mediante el uso de los cursores y esta ilustrado, dentro del rectángulos de color rojo e indicado con una flecha color azul, el tiempo es de **8,020,651pS**, observe que esta estimación considera el tiempo desde que la salida ***Page64*** se pone en un uno lógico ya que como se indica en la tabla 7.12, es un indicador cuando ya esta lista para pasar a la siguiente página o bien a la siguiente operación de las sesenta y cuatro operaciones.

Esta estimación fue comparada con la ejecución de la función ***Cap_Conv1*** que se ilustra en la Figura 8.31, dicha estimación se realizó como se ilustra en la Figura 8.39.

```
>> tic; RE1=Cap_Conv1(1); toc;
Elapsed time is 0.039783 seconds.
>>
```

8.39, Tiempo de ejecución de la función Cap_Conv1implementada en Matlab.

Como se puede observar en al Figura 8.39, se obtiene el tiempo de la función ***Cap_Conv1*** implementada en Matlab el cual es de **39.783ms**, recordando el tiempo que se obtuvo en la simulación del HW ***Cap_Conv1***, es de **8,020,651pS**, por lo podemos decir que la descripción de HW de ***Cap_Conv1*** es más rápida que la implementación de ***Cap_Conv1*** en Matlab.

8.6 Resumen de los Resultados

Se obtuvo la descripción de HW de un procesador de convocación el cual se le nombro como ***Ingrom*** y la descripción de una interfaz de imagen que permite leer imágenes con el formato BMP con un tamaño de 25×25 pixeles. Al unir estas dos descripciones de HW se obtuvo un ISP con nombre ***PI***.

Se obtuvo un ISP que permite realizar el procesamiento de un canal de una imagen con el formato BMP de un tamaño máximo de 25×25 ***pi***xeles, dicho procesamiento **lo ejecuta 217 veces más rápido que Matlab.**

Para comprobar el funcionamiento de este ISP se obtuvo la detección de bordes de una imagen mediante la simulación del ISP ***PI*** y fue comparada con la detección de bordes de la misma imagen en Matlab y los resultados fueron idénticos.

También se describió la arquitectura del HW de un SOC el cual realiza la primera capa de convolución del algoritmo propuesto por Chao Dong et. Este SOC realiza operaciones de convolución 4960 más rápido que Matlab. Sin embargo, se tienen mejores resultados en Matlab ya que los datos obtenidos por este SOC están limitados a obtener y operar con solo cuatro decimales.

Capítulo 9

Conclusiones y Trabajos Futuros

9.1 Conclusiones

9.1.1 Conclusiones generales

Se diseño y describió en HW de un ISP, el cual es una herramienta fundamental para iniciar la implementación en una FPGA de un algoritmo que resuelven el problema de SR basado en una CNN.

Se determino que esta herramienta es muy velos ya que puede realizar el procesamiento de un canal de una imagen con el formato BMP y un tamaño de 25×25 pixeles, hasta 217 veces más rápido que Matlab. Gracias a este logro se estipula que el algoritmo propuesto por Chao Dong et [46], mediante la implementación en la FPGA, obtenga la imagen de alta resolución más rápido que con su implementación en Matlab [47].

Se diseño y describió en HW un SOC que realiza la primer capa de convolución del algoritmo SRCNN propuesto por Chao Dong et [47] y en efecto se analizó que al implementar la versión de ***Ingrom_9x9*** la obtención de las operaciones de convolución para esta capa son muy veloces a comparación con la realización de la operación de convolución con Matlab, por lo que nuevamente se puede decir que al implementar herramientas en una FPGA se estipula qué la obtención de una imagen de alta resolución se adquirida más rápido que con la implementación en Matlab que propone Chao Dong et [47]. Cabe recalcar que hasta el momento este SOC solo puede realizar la extracción de un parche de una imagen con el tamaño de 10×10 pixeles.

9.1.2 Conclusiones particulares

Se realizó la investigación de un algoritmo al que se le diseñara una herramienta fundamental pala iniciar la implementación en una FPGA el cual esta basado en las CNN.

El diseño y la descripción de HW del ISP no solo es una herramienta fundamental para la implementación de una SRCNN, sino que también puede ser útil para un algoritmo de

detección de objetos basados en las CNN. Como su pudo ver en observar en los resultados la herramienta obtenida también permite realizar el procesamiento de imágenes y puede ser muy útil para diversas aplicaciones de la visión por computador.

El diseño y la descripción de HW del procesador de convolución, estipula que es una herramienta muy poderosa que la humanidad puede usar para realizar la operación de convolución para señales digitales específicas y filtros digitales específicos, ya que el proceso del diseño del procesador ***Ingrom*** se puede replicar para, señales bidimensional de mayor tamaño y además para señales unidimensionales.

El diseño y la descripción de HW de la interfaz de imagen la cual se sabe que permite extraer un canal de una imagen con el formato BMP y un tamaño máximo de 25×25 pixeles, el cual dicho canal puede ser manipulado por el procesador ***Ingrom***, sin embargo, el canal extraído puede ser manipulado por cualquier sistema que se adapte a esta descripción. La limitación de manipular solo imágenes de un tamaño máximo de 25×25 pixeles radica en la descripción de HW de las memorias donde se almacenan la extracción del canal de la imagen y el procesamiento del canal, ya que el que el *ISE* de *Xilinx* y la FPGA solo permiten el diseño de un determinado tamaño de memoria y este contexto afecto directamente a la limitación del procesamiento de la imagen que puede realizar el ISP ***PI***.

9.2 Trabajos Futuros

9.2.1 Trabajos Futuros que se pretenden desarrollar

- La implementación de ISP ***PI*** en una FPGA.

- La integración de la descripción de HW de un lector de memorias que le permita al ISP ***PI*** almacenar y extraer datos de una memoria existente en el mercado.

- Modificar el SOC ***Cap_Conv1*** para manipular completamente una imagen de tamaño de $\mathbf{105 \times 105}$ pixeles.

- La descripción de HW de un SOC que ayude a realizar la capa de convolución dos y tres, el cual como se sabe estas capas operan con kernels de tamaño $\mathbf{5 \times 5}$ por lo que este SOC debe de tener una versión de ***Ingrom*** que manipule kernels con este tamaño.

- La descripción de HW completa del algoritmo propuesto por Chao Dong et.

9.2.2 Trabajos Futuros que se proponen a desarrollar

- Se propone la fabricación del procesador ***Ingrom*** ya que se concluyó que con este se puede realizar la operación de convolución de manera eficiente, para señales digitales de un tamaño especificó.
- Se propone el uso de la herramienta diseñada y descrita en HW, para realizar la implementación de la detección de objetos basadas CNN.
- Se propone el uso de la herramienta ***PI*** diseñada y descrita en HW, para realizar el procesamiento de imágenes digitales, para aplicarla en diversas áreas de la visión por computador.
- Se propone realizar el diseño y la descripción de HW de un procesador de convolución, para señales unidimensionales, con la finalidad de utilizarlo en las diversas áreas del procesamiento de señales unidimensionales.
- Se propone el uso del procesador ***Ingrom*** en sus dos versiones para las diversas áreas del procesamiento digital de señales bidimensionales.

Referencias

[1] Aggelos K. Katsaggelos, Rafael Molina, Javier Mateos, Super Resolution of Images and Video, Morgan & Claypool, 1 jun. 2007, p. 1.

[2] E. Camargo Fernández, *Evaluación del desempeño de 2 métodos de súper resolución aplicado a imágenes médicas en 2d*, Lima Perú: universidad nacional de ingeniería facultad de ingeniería industrial y de sistemas, 2016.

[3] E. Serrano Cañizares, *SÚPER-RESOLUCIÓN EN FPGA'S PARA PROCESAMIENTO A BORDO EN SATÉLITES*, Ciudad Real: UNIVERSIDAD DE CASTILLA-LA MANCHA ESCUELA SUPERIOR DE INFORMÁTICA, 2017.

[4] R. A. Jara Pinochet, *IMPLEMENTACIÓN Y EVALUACIÓN DE ALGORITMOS DE SÚPER-RESOLUCIÓN PARA IMÁGENES TOMADAS CON NANO SATÉLITES*, Santiago: UNIVERSIDAD DE CHILE FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS DEPARTAMENTO DE INGENIERÍA ELÉCTRICA, 2018.

[5] Jung-Woo Chang y Suk-Ju Kang, «Optimización del acelerador de redes neuronales convolucionales basado en FPGA para imágenes Súper resolución,» *IEEE*, 2018.

[6] M. Marzoa Tanco, *Súper-Resolución en Imágenes*, Instituto de Computación-Facultad de Ingeniería Universidad de la Republica, 2019.

[7] Yongwoo Kim, Jae-Seok Choi y Munchurl Kim, «Una red neuronal convolucional en tiempo real para superresolución en FPGA con aplicaciones para Servicios de video 4K UHD 60 FPS,» *IEEE*, vol. 29, 8 Agosto 2019.

[8] S. Souverville Orozco, *DISEÑO DE UN ALGORITMO DE SÚPER RESOLUCIÓN EN IMÁGENES DIGITALES UTILIZANDO LA TEORÍA DE LÓGICA DIFUSA*, Ciudad de México, 2016.

[9] R. Rodríguez, *SÚPER-RESOLUCIÓN USANDO UNA ÚNICA IMAGEN*, Centro De Investigación En Matemáticas, 2016.

[10] Rafael C. Gonzalez y Richard E. Woods, Digital Image Processing, New Jersey: Pretince Hall, 2008, p. 35.

[11] E. M. i. Garcia, Visión Artificial, Barcelona, España: UOC, 2012, pp. 8-7.

[12] Esopo, «https://iie.fing.edu.uy/proyectos/esopo/eem/,» [En línea].

[13] Rafael C. Gonzalez y Richard E. Woods, Digital Image Processing, New Jersey: Pretince Hall, 2008, pp. 45-46.

[14] L. V. LABS, "LUCID VISION LABS," [Online]. Available: https://thinklucid.com/tech-briefs/understanding-digital-image-sensors/. [Accessed Abril 2021].

[15] E. M. i. Garcia, Visión Artificial, Barcelona, España: UOC, 2012, p. 12.

[16] Francisco Guindos Rojas y José Antonio Fernández, TRATAMIENTO DIGITAL DE IMAGENES CON IMtdi, Almeria, España: Universidad de Almeria. Servicio de Publicaciones, 2001, p. 11.

[17] Francisco Guindos Rojas y José Antonio Piedra Fernández, TRATAMIENTO DIGITAL DE IMAGENES CON IMtdi, Almeria, España: Universidad de Almeria. Servicio de Publicaciones, 2001, p. 12.

[18] Francisco Guindos Rojas y José Piedra Fernández, TRATAMIENTO DIGITAL DE IMAGENES CON IMtdi, Almeria, España: Universidad de Almeria. Servicio de Publicaciones, 2001, p. 13.

[19] Francisco Guindos Rojas y José Antonio Piedra Fernández, TRATAMIENTO DIGITAL DE IMÁGENES CON IMtdi, Almería, España: Universidad de Almería. Servicio de Publicaciones, 2001, pp. 14-17.

[20] C. A. Poynton, A Technical Introduction to Digital Video, New York: John Wiley & Sons, 1996, p. 175.

[21] N. Campos, «sistenix,» [En línea]. Available: https://sistenix.com/rgb2ycbcr.html. [Último acceso: Abril 2021].

[22] P. Milanfar, Super-Resolution Imaging, Boca Raton, FL: CRC Press, 2011, pp. 1-3.

[23] V. Bannore, Iterative-Interpolation Super-Resolution Image Reconstruction: A Computationally Efficient Technique, Berlin Heidelberg, Alemania: Springer, 2009, pp. 5,7.

[24] P. P. Cruz, INTELIGENCIA ARTIFICIAL CON APLICACIONES EN LA INGENIERÍA, D.F, México: Alfaomega Grupo, 2010, pp. 1-3.

[25] P. P. Cruz, INTELIGENCIA ARTIFICIAL CON APLICACIONES EN LA INGENIERÍ, D.F, México: Alfaomega Grupo, 2010, pp. 6-11.

[26] Stuart Russell y Peter Norvig, Inteligencia Artificial Un Efoque Moderno, Hoboken, Nueva Jersey, Estados Unidos: PRENTICE HALL, 2008, p. 13.

[27] F. Berzal, Redes Neuronales & Deep Learning, Granada, España: EUG, 2018, p. 344 y 349.

[28] F. Berzal, Redes Neuronales & Deep Learning, Granada, España: EUG, 2018, pp. 599,600.

[29] F. Berzal, Redes Neuronales & Deep Learning, Granada, España: EUG, 2018, p. 605.

[30] F. Berzal, Redes Neuronales & Deep Learning, Granada, España: EUG, 2018, p. 606.

[31] F. Berzal, Redes Neuronales & Deep Learning, Granada España: EUG, 2018, pp. 627,628.

[32] F. Berzal, Redes Neuronales & Deep Learning, Granda, España: EUG, 2018, p. 637.

[33] MathWorks. [En línea]. Available: https://es.mathworks.com/discovery/convolutional-neural-network-matlab.html.

[34] Intel. [En línea]. Available: https://www.intel.com/content/www/us/en/products/details/fpga/resources/overview.html.

[35] Xilinx, «Introduction to FPGA Design with Vivado High-Level Synthesis,» 2019. [En línea]. Available: https://www.xilinx.com/support/documentation/sw_manuals/ug998-vivado-intro-fpga-design-hls.pdf. [Último acceso: 29 Abril 2020].

[36] Xilinx. [En línea]. Available: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html.

[37] Chao Dong, Chen Change Loy, Kaiming He y Xiaoou Tang, «Image Super-Resolution Using Deep Convolutional Networks,» *IEEE*, vol. 38, p. 3, 1 Junio 2015.

[38] Chao Dong, Chen Change Loy, Kaiming He y Xiaoou Tang, «Image Super-Resolution Using Deep Convolutional Networks,» *IEEE*, vol. 38, p. 4, 1 Junio 2015.

[39] Chao Dong, Chen Change Loy, Kaiming He y Xiaoou Tang, «Image Super-Resolution Using Deep Convolutional Networks,» *IEEE*, vol. 38, p. 5, 1 Junio 2015.

[40] Chao Dong, Chen Change Loy, Kaiming He y Xiaoou Tang, «Image Super-Resolution Using Deep Convolutional Networks,» *IEEE*, p. 6, 1 Junio 2015.

[41] «web.archive.org,» [En línea]. Available: https://web.archive.org/web/20071026050307/http://bmrc.berkeley.edu/courseware/cs294/fall97/assignment/psnr.html. [Último acceso: Abril 2021].

[42] W. Stallings, Organización y arquitectura de conputadores, vol. Septima edición , Madrid: Pearson Educación S.A, 2005, p. 20.

[43] blogspot, «blogspot,» [En línea]. Available: http://uin14131.blogspot.com/p/h4-margin-top-0.html. [Último acceso: 9 Junio 2021].

[44] J. Miano, Compressed Image File Formats, Addison Wesley Longman, 1999, p. 23.

[45] J. Miano, Compressed Image File Formats, Addison Wesley Longman, 1999, p. 24 y 25.

[46] C. C. L. K. H. y. X. T. Chao Dong, «Image Super-Resolution Using Deep Convolutional Networks,» *IEEE*, 2015.

[47] I. S.-R. U. D. C. Networks, «mmlab,» 2015. [En línea]. Available: http://mmlab.ie.cuhk.edu.hk/projects/SRCNN.html. [Último acceso: Agosto 2021].

[48] M. d. u. d. l. Spartan-6, «Xilinx,» 2010. [En línea]. Available: https://www.xilinx.com/support/documentation/user_guides/ug384.pdf. [Último acceso: Junio 2021].

[49] T. 5. S. (I), «Universidad de Sevilla,» 2016. [En línea]. Available: http://asignatura.us.es/imagendigital/Tema5-1_SegmentacionDiscontinuidades.pdf. [Último acceso: Junio 2021].

Anexo

Cronograma de actividades