VNIVERSIDAD NACIONAL AVFNOMA DE MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

FACULTAD DE INGENIERÍA

COMPRESIÓN MJPEG DE VIDEO DIGITAL EN UN DSP

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN INGENIERÍA

ELÉCTRICA - SISTEMAS ELECTRÓNICOS

P R E S E N T A :

MARCOS DE JESÚS VITELA RODRÍGUEZ

TUTOR:

M.I. LARRY HIPÓLITO ESCOBAR SALGUERO

2006

JURADO ASIGNADO:

Presidente: Dr. Francisco García Ugalde

Secretario: Dr. Felipe Orduña Bustamante

Vocal: M.I. Larry Hipolito Escobar Salguero

1er Suplente: Dr. Boris Escalante Ramírez

2do Suplente: Dr. Jesús Savage Carmona

Lugar o lugares donde realizó la tesis

Posgrado, Facultad de Ingeniería U.N.A.M

Tutor de Tesis

M.I. Larry Hipolito Escobar Salguero

_____

Firma

¡El día que la mierda tenga algún valor, los pobres nacerán sin culo!

Gabriel García Márquez

¿Que pasa si el emperador Cuauhtémoc se baja de su pedestal en Reforma y empieza a caminar entre la gente? ¡Pues que le vuelven a quemar las patas!

Carlos Fuentes

Vine a Comala porque me dijeron que acá vivía mi padre un ¡tal Pedro Páramo!

Juan Rulfo

No queremos que el Gobierno nos "eche la mano", sino que "nos quite el pie de encima"

El pueblo jodido

A toda mi familia, en especial a mis padres

¡No existen palabras para describir mi agradecimiento!

Agradecimientos

A la Universidad Nacional Autónoma de México y Instituto Tecnológico de la Laguna por haberme permitido formar parte de ellos y brindarme las herramientas necesarias durante mis estudios.

*Al Consejo Nacional de Ciencia y Tecnología (**CONACYT**) y la Dirección General de Estudios de Posgrado (**DGEP**) por concederme las becas de apoyo y de esta forma poder realizar y concluir satisfactoriamente los estudios de maestría.*

*Un reconocimiento especial al **M.I. Larry Hipólito Escobar Salguero** por contribuir con su conocimiento y experiencia en la realización de esta tesis, así como también por la orientación, apoyo, amistad y regaños recibidos en todo momento.*

Al Dr Francisco García Ugalde, Dr. Felipe Orduña Bustamante, Dr. Boris Escalante Ramírez y al Dr. Jesús Savage Carmona por sus comentarios y sugerencias para complementar esta tesis.

Un agradecimiento final para todas las personas que me brindaron su apoyo y amistad en todo momento.

Así mismo mi desprecio más absoluto para todas aquellas personas que con sus palabras dicen defender al pueblo de México y con sus hechos lo traicionan y apuñalan, Malditos y Despreciables sean pues Vicente Fox Quesada y Felipe Calderón Hinojosa.

ÍNDICE GENERAL

CAPITULO III.- EL ESTÁNDAR DE COMPRESIÓN PARA VÍDEO MJPEG

CAPITULO IV.- IMPLEMENTACIÓN DEL ESTÁNDAR MJPEG EN UN DSP

RESUMEN

La ciencia y la tecnología de las comunicaciones han hecho posible la difusión de video a través de grandes distancias, teniendo como consecuencia que las imágenes puedan ser percibidas por la gran mayoría de la población mundial. Los métodos que hacen posible este logro tan significativo son amplios y muy variados consecuencia de múltiples aportaciones de una gran cantidad de personas.

A pesar de los avances tecnológicos, para trasmitir o almacenar señales digitales de video, en aplicaciones como *INTERNET*, es necesario efectuar procesos de compresión que nos permitan representar señales de video con una cantidad menor de información que las señales originales, tratando de no afectar demasiado su contenido visual. Estos procesos de Compresión son necesarios, debido a la gran cantidad de información para representar una señal de video.

El siguiente trabajo tiene como finalidad abordar una parte del área de Compresión de imágenes en movimiento que en la actualidad se realiza.

El objetivo de este trabajo es realizar un *"codificador – decodificador "* de vídeo digital utilizando el estándar *MJPEG* (Unión Grupal de expertos de imágenes en movimiento) en un *Procesador Digital de Señales* (*DSP)*; los algoritmos *MJPEG* tienen su base en la *Transformada Coseno Discreta (DCT*), la cual es una herramienta de cálculos intensivos. El algoritmo de Compresión *MJPEG* implementado en la arquitectura del *DSP*, nos permitió obtener porcentajes de compresión entre 1 - 24%, con respecto a las secuencias originales, así mismo el algoritmo utilizado para la decompresión *MJPEG*, nos entregó diferentes calidades visuales para las secuencias de video reconstruidas.

La finalidad de aplicar el estándar *MJPEG* en la arquitectura del *DSP*, es evaluar diferentes grados de compresión con respecto a sus calidades visuales, así como obtener los tiempos de ejecución tanto para la codificación como decodificación, para concluir si es factible su implementación en tiempo real en este tipo de arquitectura.

INTRODUCCIÓN

El ser humano tiene la innata necesidad de estar comunicado siendo esencial el estar informado para mejorar el entendimiento de su entorno, una de las formas mas importantes para interactuar con el medio ambiente que le rodea es a través de sus sentidos *Audio-Visuales.*

Con el surgimiento de la electrónica y las comunicaciones se ha hecho posible manipular, procesar, almacenar y transmitir información de imágenes y sonido gracias a su representación a través de señales eléctricas de variación continua; sin embargo el nacimiento de las computadoras y dispositivos digitales trajo consigo una manera distinta de representar la información, en ellas las señales eléctricas toman solo dos niveles posibles.

Así mismo la representación digital de información tiene ventajas con respecto a la analógica como son: mayor calidad, menor ruido, permite una mayor manipulación de las señales, a su vez los datos de tipo digital pueden ser replicados exactamente y distribuidos fácilmente lo cual les da mucha mayor flexibilidad con respecto a los datos en formato analógico. Este tipo de ventajas han mejorado una gran cantidad de productos ya existentes, a su vez han contribuido a crear muchos otros, tanto para uso doméstico como en aplicaciones especializadas.

Uno de los avances que ha hecho posible la llamada *"revolución digital"* ha sucedido en la tecnología de semiconductores generando dispositivos de Silicio que son mas pequeños, poderosos, flexibles y baratos. En ellos las funciones como microprocesamiento, lógica, procesamiento de señales y memoria están integrados sobre un mismo componente, lo cual proporciona varias ventajas, como el menor consumo de potencia, reducción en los costos de fabricación, además de poder ser usados para múltiples aplicaciones y productos dependiendo del *Software* almacenado sobre ellos.

Estas aplicaciones han aumentado las necesidades de procesamiento para los datos de tipo digital, como voz, audio, video, etc. Para lograr un procesamiento eficaz de estos datos, se requiere de un dispositivo extremadamente veloz; uno de los dispositivos que apareció para el mejoramiento del procesamiento digital de señales *(PDS)* fue el procesador digital de señales *(DSP)* cuya función principal es el efectuar operaciones de manera rápida mediante el diseño de su arquitectura.

A su vez no solo es importante el contar con dispositivos que sean capaces de procesar información de manera rápida, si no también es importante contar con técnicas que mejoren el tratamiento de la información que generan las señales digitales. Dentro de estos procesos está la *compresión de datos*, que optimiza la representación que ocurre en los sistemas digitales.

La *compresión de datos* y en especial la de imágenes resulta cada vez mas importante, debido al rápido desarrollo de las computadoras, al crecimiento de los sistemas *multimedia* y al gran avance de *INTERNET*, la compresión de imágenes juega un papel fundamental en muchas y diversas aplicaciones que dependen del procesamiento, almacenamiento y transmisión de imágenes digitales.

El objetivo de este trabajo es realizar la implementación un codificador-decodificador *MJPEG* sobre una arquitectura tipo *DSP*. El sistema a realizar tiene como partes principales una computadora PC y la tarjeta de desarrollo *DSK6711,* que es donde se encuentra localizado el *DSP,* esta tarjeta cuenta con los componentes adecuados para comunicarse con la PC a través de un puerto paralelo *LPT*, además de contar con la suficiente memoria para el almacenamiento de los datos.

La programación junto a la secuencia de video se descargará de la PC hacía el DSK donde se almacenará para ser procesada, una vez que sea codificada se procederá a realizar su decodificación, al tener de nuevo la secuencia recuperada esta se descargará hacía la PC donde se desplegará por medio del software adecuado (MATLAB).

La finalidad de aplicar el estándar en la arquitectura del *DSP* es evaluar diferentes grados de compresión con respecto a sus calidades visuales, así como obtener los tiempos de ejecución tanto para la codificación como decodificación, para concluir si es factible su implementación en tiempo real en este tipo de arquitectura.

Este trabajo está estructurado de la siguiente manera:

El ***capitulo I*** presenta los conceptos fundamentales acerca de las imágenes digitales, así como del video analógico y digital.

El ***capitulo II*** proporciona una descripción general acerca de la teoría de la compresión de video, así como de sus principales estándares entre los cuales se pueden citar: H.261 y MPEG1.

En el ***capitulo III*** se realiza una descripción detallada acerca del estándar utilizado en este trabajo (*MJPEG*).

El ***capitulo IV*** expone como se realizó la implementación en la arquitectura del DSP.

En el ***capitulo V*** por último se muestran los resultados obtenidos para diferentes grados de compresión aplicados.

Adicionalmente la documentación cuenta con secciones que hacen referencia a ***Conclusiones, Anexos, Glosario*** y citas ***bibliográficas***

CAPITULO I.- IMAGEN Y VÍDEO DIGITAL

Este capitulo tiene como objetivo el describir, los principales conceptos acerca del video digital, así mismo se da un breve repaso acerca del video analógico lo cual permite una mejor compresión de su contraparte digital.

1.1.- Imágenes

De acuerdo a algunas definiciones provenientes del diccionario el término imagen significa: copia, figura, dibujo, fotografía, grabado, ilustración, representación mental de un objeto, siendo correctas todas ellas [16]. Para obtener una imagen no es necesario percibir directamente el fenómeno a representar, por ejemplo el desarrollo tecnológico de las últimas décadas ha permitido la generación de imágenes empleando radiación invisible a la visión humana, imágenes acústicas, magnéticas, de radar [9].

Sin embargo las imágenes más importantes para el ser humano son las imágenes ópticas las cuales pueden percibirse directamente por el ojo humano, éstas se pueden clasificar de muchas maneras pudiéndose ser imágenes fijas, en movimiento, continuas o discretas por mencionar solo algunas de sus características. Una definición mas rigurosa de imágenes continuas y discretas es la siguiente [6]:

Una imagen continua es aquella donde la variación de tonos de gris o color se presenta sin discontinuidades, sin líneas o fronteras aparte de las que pudiera tener la escena misma, una imagen discreta por su parte es la que está compuesta por elementos definidos y diferenciados como puntos o cuadrados.

1.1.1.- Imagen digital

Para nuestro trabajo nos interesan solamente las imágenes discretas, como un subconjunto de ellas se pueden encontrar las *imágenes digitales*; el hecho de que una imagen sea digital implica que los elementos que la forman solo podrán tener valores formados por las combinaciones de 0 y 1. Al digitalizar una imagen, se produce una pérdida de información con respecto a la imagen continua.

El hecho de que la información contenida en una imagen digital, sean combinaciones de unos y ceros permite que se pueda hacer referencia a cualquier cosa, de ahí que la información numérica de una imagen almacenada en un archivo pueda ser teóricamente idéntica a la de un sonido o un texto [9].

1.1.1.1.- Tipos básicos de imágenes digitales

Puesto que la información digital es discontinua toda imagen de este tipo ha de estar dividida en unidades claramente identificables que contengan cada una un conjunto de información determinada, ha este respecto existen dos tipos de imágenes digitales.

- Las creadas mediante porciones gráficas de la imagen*.*
- Las creadas mediante elementos definidos matemáticamente.

A las primeras se les denomina imágenes de *mapas de bits* a las segundas *imágenes vectoriales* [9].

1.1.2.- Imágenes de mapas de bits

Las imágenes de mapas de bits (bitmaps o imágenes raster) están formadas por una rejilla de celdas a cada una de las cuales se les denomina píxel (elemento de imagen por sus siglas en ingles), a dichos elementos se les asigna un valor propio (dependiendo del modo de color utilizado) de tal forma que su agrupación crea la ilusión de una imagen en tono continuo.

Los píxeles son unidades de información mas no de medida, significando que contienen información independientemente de su tamaño, por ejemplo un píxel puede ser muy pequeño (0.1mm) o muy grande (1 m).

Una imagen de mapa de bits es creada mediante una rejilla de píxeles única cuando se modifica su tamaño, se modifican grupos de píxeles, no los objetos o figuras que contiene, por lo que estos suelen deformarse o perder algunos de los píxeles que los definen; por lo tanto una imagen de bits está diseñada para un tamaño determinado perdiendo calidad si se modifican sus dimensiones [9].

1.1.2.1.- Resolución

La resolución se define como el número de píxeles que tiene una imagen por unidad de longitud, es decir la densidad de píxeles en la imagen, una forma común de clasificar imágenes según su resolución es aquella que las divide en imágenes de *alta resolución* e imágenes de *baja resolución*.

A mayor resolución existen más píxeles en una imagen y por lo tanto su mapa de bits es mas grande, contiene mayor información y es mayor su capacidad de distinguir los detalles espaciales finos por lo que tendrá mas definición, permitiendo transiciones de color mas suaves y una mayor calidad de reproducción [9].

1.1.2.2.- Profundidad de bits

Como parte de la información que contiene un píxel para representar la imagen original se le asigna una cantidad determinada de bits, ha esta cantidad se le denomina profundidad de bits. Se trata de un concepto importante porque a mayor profundidad de bits mas información contiene la imagen y por consiguiente se puede tener un mayor número de colores.

- Si la profundidad es de un solo bit solo existe la posibilidad de tener *dos niveles o tonos.*
- Si la profundidad es de dos bits es posible tener *cuatro niveles o tonos*.

Los niveles que podrá contener una imagen se encuentran mediante la siguiente relación 2^L, siendo ***L*** el número de profundidad de bits; para imágenes en tono real se tiene una profundidad de 24 bits generalmente lo cual genera ***16,777,216*** colores posibles para su representación [7].

1.1.2.3.- Peso

Los mapas de bits pueden estar definidos en un número variable de colores, cuanto más colores tenga la imagen, mayor calidad tendrá, así mismo si su resolución es elevada, mayor *peso* requerirá para su almacenamiento. El *peso* de una imagen digital sin ninguna técnica de compresión aplicada se define en la ecuación **(1.1)** como [9]:

$$\frac{(\text{Resolución ancho} \times \text{Resolución alto}) \times \text{profundidad bits}}{8192} \qquad \textbf{(1.1)}$$

La ecuación **(1.1)** proporciona el peso de la imagen en Kbytes, como ejemplo se tiene una imagen con resolución de 256 píxeles x 256 píxeles, con una profundidad de bits de 24 (imagen en tono continuo) su peso es:

$$((256 \times 256) \times 24) / 8192 = 192 \text{ Kbytes}$$

1.1.2.4.- Frecuencia espacial

Un concepto íntimamente ligado con las imágenes es el de frecuencia espacial, para su comprensión es necesario abordar el concepto de frecuencia temporal, en general las señales unidimensionales de alta frecuencia cambian su valor en un período corto de tiempo, si se supone una señal senoidal con periodo 2π y otra con periodo π, está última con menor periodo cambia más rápidamente que la primera es decir tiene mayor frecuencia.

Si se considera una imagen como la señal a tratar, se define que una imagen con alta frecuencia espacial, cambia el valor de los niveles de intensidad en un intervalo pequeño, o lo que es lo mismo en distancias pequeñas de la imagen, el resultado visual es que los niveles de intensidad cambian de forma abrupta de un nivel a otro, por el contrario las bajas frecuencias corresponden a cambios mas lentos en la variación de los niveles, donde los cambios ocurren gradualmente de una posición a otra de la imagen [8].

1.1.2.5.- Espacios de color

Desde que Sir Isaac Newton descubriera la descomposición de la luz en el espectro se han propuesto numerosos modelos sobre qué es el color, el más importante de todos ellos, y que aún sirve de fundamento a la mayoría de los sistemas de color usados en la actualidad fue el de Thomas Young, que en el siglo XVIII propuso que el ojo creaba todos los matices mediante una mezcla de tres colores básicos; esta hipótesis es conocida como teoría tricromática de la luz [7].

La teoría tricromática señala que idealmente tres arreglos de muestras (tres componentes) deben ser suficientes para representar una imagen en color. El sistema RGB (rojo, verde y azul) es un ejemplo de la representación en color que requiere tres valores independientes para describir los colores, cada uno de los valores puede ser variado independientemente y por lo tanto se puede crear un espacio tridimensional con las tres componentes. Como coordenadas independientes, los colores son representados como puntos en este espacio los tonos de gris desde el negro hasta el blanco se encuentran en la línea diagonal de la ***Figura 1.1*** [1].

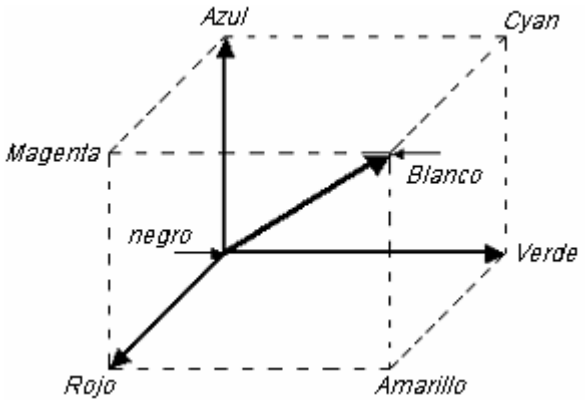


Figura 1.1.- Espacio coordenado de color RGB

1.1.2.6.- Transformaciones lineales del color

Las representaciones tales como RGB no siempre son las mas convenientes, otras representaciones que usan componentes de color están relacionadas directamente con la forma visual de percepción.

Los espacios de color o sistemas de color coordenado tienen una componente de luminancia y los otros dos de crominancia, la luminancia provee una versión en escala de grises de la imagen y los componentes de crominancia proveen la información extra que convierte la imagen en escala de grises a una imagen en color [1].

El sistema visual de luminancia es la suma de diferentes porciones de los componentes R, G, B por conveniencia los valores de estos tres colores son expresados por una escala relativa de 0 a 1, donde 0 indica que no existe excitación y 1 indica máxima excitación; la luminancia ***Y*** puede ser calculada por la ecuación **(1.2)** [1].

$$Y = 0.3\ R + 0.6\ G + 0.1\ B \quad (1.2)$$

El término crominancia está definido como la diferencia entre un color y la luminancia, la información de crominancia puede ser expresada por un conjunto de diferencias de color ***Cb*** e ***Cr*** que se definen de acuerdo a las ecuaciones **(1.3)** e **(1.4)** respectivamente [1].

$$Cr = ((R - Y) / 1.6) + 0.5 \quad (1.3)$$

$$Cb = ((B - Y) / 2) + 0.5 \quad (1.4)$$

Estas diferencias de color son cero si R = G = B lo que produce diferentes niveles de gris, que no contienen crominancia; junto con la luminancia estas coordenadas forman el sistema de color conocido como ***YCrCb***. Otro espacio de color ***YIQ*** es usado en algunos sistemas de TV, el sistema ***YIQ*** presenta la ecuación **(1.2)** para su componente de luminancia, pero las ecuaciones para ***I*** e ***Q*** son definidas en **(1.5)** y **(1.6)**.

$$I = 0.6\ R - 0.282\ G - 0.317\ B \quad (1.5)$$

$$Q = 0.213\ R - 0.534\ G + 0.321\ B \quad (1.6)$$

Independientemente del modo de color utilizado en la representación de la imagen existen algunos conceptos importantes relacionados con su tratamiento [7].

- ***Brillo:*** Es la intensidad de iluminación total o promedio que determínale nivel de fondo en la imagen reproducida.

- ***Contraste:*** Es la diferencia de intensidad entre las partes blancas y las negras de la imagen reproducida. El intervalo de contraste debe ser tan amplio como para producir una imagen intensa, con blanco brillante y negro oscuro para valores extremos.

- ***Saturación:*** Los colores saturados son brillantes, intensos o fuertes, los colores pálidos o débiles tiene poca saturación; la saturación indica como está diluido el color por el blanco; por ejemplo el rojo brillante o intenso está completamente saturado cuando éste se diluye por el blanco, el resultado es el rosa que en realidad es un rojo no saturado.

- ***Tinte:*** De manera mas específica el color de un objeto se denomina tinte o matiz, las hojas verdes tienen matiz verde una manzana roja tiene matiz rojo.

1.1.3.- Imágenes vectoriales

Las imágenes vectoriales también son conocidas como gráficos orientados a objetos, es el segundo grupo de imágenes digitales y son mas simples que los mapas de bits, ya que se almacenan y representan por medio de trazos geométricos controlados por cálculos y fórmulas matemáticas, éstas no se construyen píxel a píxel, sino que se construyen a partir de vectores objetos formados por una serie de puntos y líneas rectas o curvas definidas matemáticamente.

Las principales ventajas que ofrecen las imágenes vectoriales derivadas de su naturaleza matemática son:

- Almacenan las imágenes en archivos muy compactos ya que solo se requiere la información necesaria para generar cada uno de los vectores, dado que no se ha de almacenar información para definir cada píxel de la pantalla, sino una serie de fórmulas matemáticas.

- Permiten modificar el tamaño de las imágenes y de sus objetos sin que se produzca pérdida de información ya que se analizan de forma matemática todas las nuevas relaciones y posiciones de los elementos [9].

1.2.- Video

Una imagen "fija" es una distribución espacial de intensidad que es constante con respecto al tiempo, por otro lado el vídeo es un patrón de intensidad espacial que cambia con respecto al tiempo, otro significado común para video es el definirlo como un conjunto o secuencia de imágenes con alto grado de relación entre si, por tanto el video se puede considerar como una secuencia de imágenes fijas [2].

1.2.1.- Vídeo Analógico

En una señal de video la amplitud del voltaje o de la corriente cambia respecto al tiempo (como en una señal de audio), las variaciones en la señal de video corresponden a información visual, producida por una cámara que convierte la luz en señales eléctricas [14].

El video tradicionalmente ha sido capturado, almacenado y transmitido en forma analógica, el término señal de video análoga hace referencia a una señal eléctrica unidimensional de tiempo que es obtenida por el muestreo del patrón de intensidad en las coordenadas horizontal, vertical y temporal y convertido en una representación de tipo eléctrica, este proceso de muestreo es conocido como barrido [2].

El barrido de una imagen comienza en la esquina superior izquierda y continua horizontalmente, cuando alcanza el final de la línea regresa al principio solo que ahora salta a la siguiente línea para comenzar de nuevo; una vez que alcanza el final de la última línea se ha completado un cuadro y el barrido regresa de nuevo a la esquina superior izquierda para comenzar a trazar el siguiente cuadro, durante el trazo son insertados los pulsos de sincronización y de borrado en la señal.

Los métodos comúnmente utilizados generan un barrido *progresivo* o *entrelazado*; en el *progresivo* un cuadro es formado por un simple barrido continuo, en el método *entrelazado* un cuadro es formado por dos barridos sucesivos denominados *campos*, en el primer campo las líneas impares son barridas formando la mitad de un cuadro, luego las líneas pares son barridas para formar el segundo campo, cuando se entrelazan las líneas los dos campos forman un cuadro.

Aunque parezca que las imágenes reproducidas por una señal de video se mueven, en realidad se muestran una serie sucesiva de imágenes fijas con suficiente rápidez para dar la ilusión de movimiento al espectador.

Líneas por segundo: el número de líneas barridas para una imagen completa debe ser grande, con la finalidad de incluir el mayor número de elementos de imagen y por tanto más detalle, existen diferentes estándares con diferente resolución, en uno de ellos la cantidad de líneas posibles por cuadro es 525 para formar la imagen completa.

Cuadros por segundo: como se mencionó el barrido horizontal se realiza en forma lenta hacia abajo, mientras se realiza el barrido horizontal también se efectúa un barrido vertical, este movimiento es necesario para no barrer las líneas una encima de la otra; el barrido horizontal produce líneas de izquierda a derecha mientras que el vertical distribuye las líneas para completar el cuadro de abajo hacia arriba.

Un cuadro está formado por 720 píxeles por línea y posee 525 líneas. Estas 525 líneas se exploran en 1/29.97 segundos, para crear la ilusión de movimiento deben mostrarse las imágenes completas suficientes durante cada segundo, este efecto se logra si se tiene una tasa de repetición de imágenes mayor que 16 por segundo. La tasa de repetición de 24 imágenes por segundo empleada en las películas de cine es suficiente para producir la ilusión de movimiento en la pantalla.

Para eliminar posibles efectos de parpadeo en el sistema cada cuadro se repite dos veces, esto se logra entrelazando las líneas de barrido horizontales en los dos campos. La tasa de repetición de los campos es 60 por segundo porque se barren dos campos durante un periodo de cuadro de 1/30 seg de esta manera se muestran 60 vistas de la imagen durante un segundo, esta tasa de repetición es suficientemente rápida para eliminar cualquier efecto de parpadeo.

La frecuencia de barrido vertical es de 60 Hz, y es la rapidez del trazo de vídeo en que completa sus ciclos de movimiento vertical de arriba abajo y luego de regreso hacia arriba. El número de líneas de barrido horizontales en un *campo* es la mitad del total de 525 líneas para un cuadro completo, como un *campo* contiene líneas alternadas se obtienen 262.5 líneas horizontales para cada campo vertical; y el tiempo para un campo es de 1/60 seg, entonces el número de líneas por segundo es

262.5 x 60 Hz = 15750 Hz

O bien si se consideran 525 líneas para un par sucesivo de campos se puede multiplicar la rapidez de cuadros de 30 por 525 que da el mismo resultado de 15750 líneas barridas en 1 segundo. Esta frecuencia de 15750 Hz es la velocidad a la cual el trazo de barrido completa sus ciclos de movimiento horizontal de izquierda a derecha y de regreso a la izquierda. El tiempo para cada línea de barrido horizontal es 1 / 15750 segundos, lo cual en términos de microsegundos es :

Tiempo H = 1 / 15750 = 63.5 µseg

El tiempo en µs indica que la señal de video para los elementos de imagen dentro de una línea horizontal puede tener altas frecuencias del orden de MegaHertz, si hubiera más líneas el tiempo de barrido sería menor y se obtendrían frecuencias de video mayores, por lo tanto la mayor frecuencia de video está limitada a cerca de 4.2 MHz [14].

1.2.1.1.- Análisis de la señal de video

Las tres partes principales de la señal de video compuesto son:

1) Señal de cámara correspondiente a las variaciones de luz en la escena.

2) Pulsos de sincronización o sincronía para el barrido.

3) Pulsos de borrado que hacen invisibles los retornos.

La señal de cámara se combina con el pulso de borrado luego se añade la sincronización para producir la señal de video compuesta de la ***Figura 1.2***.

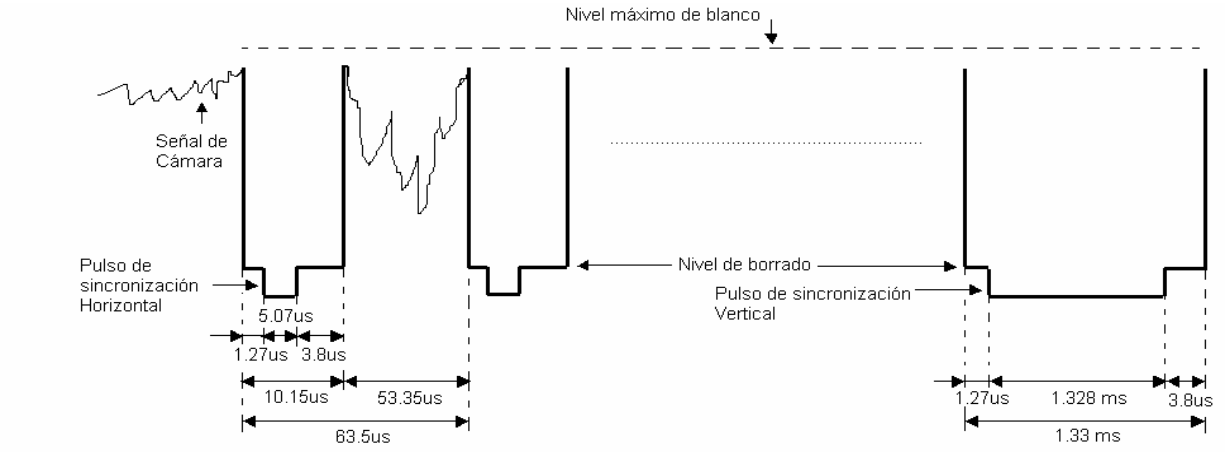


Figura 1.2.- Señal de video compuesto analógica

En la ***Figura 1.2*** se muestran los valores sucesivos de amplitud de voltaje para el barrido de dos líneas horizontales en la imagen, cuando aumenta el tiempo en la dirección horizontal las amplitudes varían para los matices de blanco, gris y negro en la imagen empezando en el extremo izquierdo. La señal está en un nivel de blanco para la primera línea en la segunda estas variaciones son más marcadas hacia el gris y negro, esté proceso continua trazando todas las líneas hasta el final del cuadro donde se genera el pulso de sincronización vertical.

El estándar de la industria para la distribución de video de las señales de entrada y salida de un equipo es una señal como la mostrada en la ***Figura 1.2*** con amplitud pico a pico de 1V, la lista del equipo que la utiliza incluye cámaras, camcorders, videocaseteras, equipos de edición, monitores receptores de TV [14].

1.2.1.2.- Sincronización

Los pulsos de sincronización se añaden como parte de la señal de video completa, un pulso de sincronización horizontal al final de cada línea determina el inicio del retorno horizontal, el retorno horizontal de barrido comienza en el lado derecho de la imagen.

La sincronización vertical al final de cada campo determina el inicio del retorno vertical. Si la sincronización de campo vertical no estuviera presente, la imagen reproducida no se mantiene fija, en el sentido vertical se movería de arriba ha abajo en la pantalla del tubo de imagen. Sin los pulsos horizontales la imagen no se detiene en el sentido horizontal, se desliza a la izquierda o a la derecha y después se separa en segmentos diagonales.

La frecuencia de barrido horizontal es 15750 Hz por lo tanto la frecuencia de sincronía horizontal también es 15750 Hz, la velocidad de repetición de los cuadros es 30 seg, pero la frecuencia de barrido vertical de campos es 60 Hz, debido a esto la frecuencia de los pulsos de sincronización vertical también es 60 Hz [14].

1.2.1.3.- Borrado

El término borrado significa *pasar a negro,* es decir, ennegrecer la pantalla como parte de la señal de video, el voltaje de borrado se encuentra en el nivel de negro, la señal de video compuesta contiene los pulsos de borrado para hacer invisibles las líneas de retorno cambiando la amplitud de la señal a negro, cuando se producen los retornos toda la información de la imagen queda suprimida durante el tiempo de borrado.

El tiempo necesario para el borrado horizontal es aproximadamente 16% del tiempo de cada línea horizontal (H), el tiempo H total es 63.5μs que incluye el trazo y el retorno, el tiempo de borrado para cada línea es entonces 63.5μs x 0.16 = 10.15 μs, este tiempo de borrado H significa que el retorno de derecha a izquierda debe hacerse en 10.15 μs antes del inicio de la información visible de la nueva línea, durante el barrido de izquierda a derecha.

El tiempo para el borrado vertical (V) se aproxima a 8% del tiempo de cada campo vertical (V), el tiempo total vertical es 1/60 seg que incluye el trazo hacia abajo y el retorno hacia arriba entonces el tiempo de barrido de cada campo es 1/60 x 0.08 = 0.0013 seg (1.3 ms) en este tiempo el retorno vertical debe completarse de abajo hacia arriba de la imagen; todos estos tiempos aparecen en la ***Figura 1.2*** [14].

1.2.1.4.- Adición del color

El análisis descrito anteriormente es para una señal en tonos de gris (*Y*), para una señal de video en color se agregan la señal de crominancia (modulada en cuadratura) de 3.58 MHz junto con una señal de ráfaga de sincronización de color, el intentar describir a fondo la realización de este proceso queda fuera de este estudio si se desea más información puede consultar las referencias señaladas [14].

1.2.1.5.- Formatos de video NTSC, PAL, SECAM

Existen diferentes formatos de vídeo compuesto tales como *NTSC* (comité nacional de sistemas de televisión por sus siglas en inglés), *PAL* (Alternación línea Fase) y *SECAM* (Systeme Electronique Color Avec Memorie) usados en diferentes países alrededor del mundo. El estándar de video compuesto *NTSC* definido en 1952 es usada principalmente en Norteamérica y Japón, la señal *NTSC* es una señal de video entrelazado con 262.5 líneas por campo (525 por cuadro), 60 campos por segundo, con una resolución horizontal de 720 elementos de imagen (también denominados *píxeles* aunque estos no están cuantizados por lo tanto no son digitales).

PAL y *SECAM* desarrollados en la década de los 60 son comúnmente utilizados en Europa, también es video entrelazado pero tiene una resolución temporal diferente en comparación con *NTSC*, además de tratar al color de una manera distinta. Ambos *PAL* y *SECAM* tienen 625 líneas por cuadro resolución horizontal de 720 y 50 campos por segundo, por lo tanto tienen una resolución vertical mayor que *NTSC*, estos parámetros generan una frecuencia de barrido de 15625 Hz; una diferencia entre *PAL* y *SECAM* es en la manera de representación del color, ambos representan mejor la información que *NTSC* [14].

1.2.1.6.- Formatos de video analógico

Los formatos de señal compuesta usualmente generan errores en el rendimiento del color, tales como errores en la saturación del color, debido a las imprecisiones en la separación de la señal de color. Existen varios estándares de señales de video analógico los cuales difieren en la forma de representar el color, los mas importantes son [14]:

- Vídeo compuesto (*PAL, SECAM, NTSC*) descrito en las ***secciones 1.2.1*** a ***1.2.1.5***.

- Vídeo componente: La señal de video es dividida en tres partes, pueden ser señales RGB, luminancia-crominancia o una transformación de ellos, este formato permite la mejor representación del color porque utiliza componentes separados.

- S-video: Es un compromiso entre el video compuesto y el video de componentes, donde se representa el vídeo con dos señales una de luminancia y una señal compuesta de crominancía que puede estar basada en la representación (I,Q) o (U,V) de los sistemas *NTSC, PAL* y *SECAM*, S-video es comúnmente usado en grabadoras de videocasete y camcorders para obtener mejor calidad que la del vídeo compuesto.

1.3.- Video digital

El video digital es simplemente un medio alternativo de representar la información contenida en una *forma de onda* de video analógica, utilizando las herramientas que nos proporciona la tecnología digital.

El proceso de digitalización de vídeo involucra tres operaciones básicas: filtrado, muestreo y cuantización; si la frecuencia de muestreo no es el doble que la frecuencia máxima de la señal analógica ocurrirán efectos de *aliasing*, por lo tanto la operación de filtrado es usada para limitar el ancho de banda de la señal de entrada; la señal analógica filtrada es muestreada en un número especifico de veces para generar una señal en tiempo discreto la mínima razón de muestreo es conocida como la razón de *Nyquist*. Por ejemplo para un sistema PAL el intervalo es de 10 – 11 MHz con lo cual la frecuencia de muestreo normalmente es de 13.5 MHz.

Las muestras resultantes del proceso de muestreo tienen amplitudes continuas, por lo tanto se requerirá precisión infinita para representarlos; la operación de cuantización es usada para mapear tales valores hacía un conjunto finito de amplitudes discretas que pueden ser representados por un número finito de bits. Cada muestra es definida como un elemento de imagen y es organizado en un arreglo bidimensional para formar una imagen digital fija o cuadro digital; por lo tanto el video digital consiste de una secuencia de imágenes digitales fijas (bitmaps).

Para producir imágenes en movimiento es necesario proporcionar un mecanismo donde el valor de cada píxel pueda actualizarse periódicamente esto da como resultado una matriz tridimensional donde dos de los ejes son espaciales y el tercero es temporal esto se muestra en la ***Figura 1.3***, en ella se aprecia como el video es una sucesión de imágenes bidimensionales que ocurren en instantes de tiempo discreto [15].

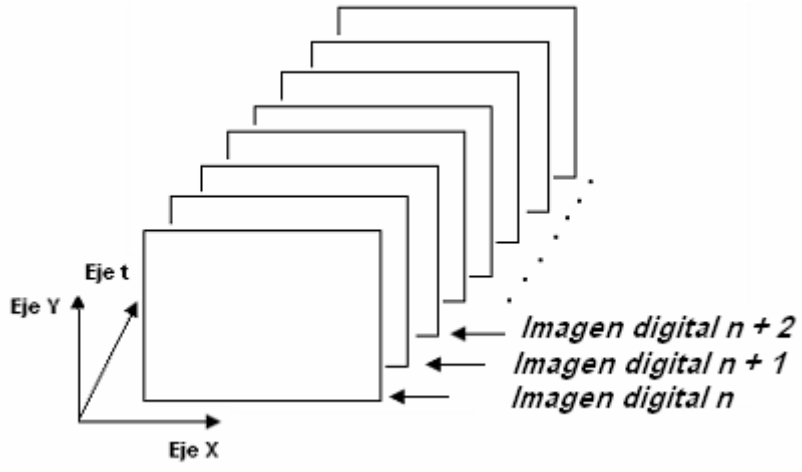


Figura 1.3 **.-** Video digital

Si la señal de video digital se encuentra en color entonces está será representada como en la figura anterior solo que cada imagen tendrá asociada sus respectivos componentes para el sistema de color que utilizen (*RGB*, *YCrCb*, etc). En el video digital no existe la necesidad de utilizar pulsos de sincronía y borrado por lo que la computadora conoce exactamente donde empieza una nueva línea, su tamaño y el número de píxeles, por lo tanto estos pulsos son removidos en la conversión A/D. Considerando como ejemplo el estándar Europeo de TV que difiere para el Norteamericano, pero el número de sus muestras por línea (píxeles) es 720 para ambos, dado que en el video digital sólo interesan las partes activas de la imagen y el número de líneas activas por cuadro es de 625, el numero total de píxeles por segundo llega a ser igual a **720 x 625 x 25 = 11, 250, 000 píxeles/s**.

La razón de bit total es calculada considerando los diferentes componentes de la imagen así como su profundidad de bits, suponiendo componentes RGB y una profundidad de 8 bits se tiene **11, 250, 000 x 3 x 8 = 270, 000, 000 bits/s** ó **257.49 Mbits/s** lo cual es una razón muy elevada para su ancho de banda, esta es la mayor desventaja del video digital en la actualidad tanto para su almacenamiento como para su transmisión [15].

1.3.1.- Ventajas del video digital

Las aplicaciones del video digital se están volviendo cada vez mas importantes, entre las principales se pueden citar, videotelefonía, videoconferencia, TV digital, TV digital de alta definición (HDTV) , aprendizaje a distancia, vigilancia, etc [8].

Las principales ventajas del video digital sobre el analógico son [2]:

- Fácil de editar, mejoras y creación de efectos especiales.
- Evita las fallas típicas del video analógico como por ejemplo aquellas causadas por la grabación repetida de las cintas, errores en la representación del color debido a las inexactitudes en la separación de señales de video compuesto.
- Fácil conversión de formatos por medio de Software, para video analógico la conversión necesita costosos equipos.
- Robustez al ruido y facilidad para los procesos de encriptación.
- Interactividad, fácil de indexar, buscar y recuperar para video analógico se requiere de tediosos escaneos.

Estas ventajas permiten un número de nuevas aplicaciones y servicios para ser introducidos, como por ejemplo la interactividad producida en la HDTV.

1.3.2.- Formatos de vídeo digital

Debido al muestreo, una señal de video alcanza una gran tasa de datos, como se vio anteriormente la tasa generada por un sistema PAL es **257.49 Mbits/s**, tal cantidad de datos imposibilitaría la aplicación del video digital en las aplicaciones de consumo (tales como camcorders digitales, TV digital). Una solución a este problema es reducir el número de componentes para las que el ojo humano tiene poca sensibilidad y a las técnicas de compresión.

Dependiendo de la aplicación, las imágenes tienen diferentes grados de resolución, por ejemplo en videoconferencia o video teléfono las imágenes son de tamaño pequeño con resoluciones bajas y por tanto requieren mucho menor ancho de banda para su transmisión. A su vez el intercambio de video digital entre diferentes industrias, redes y plataformas de Hardware requiere de formatos de video estandarizados.

Los patrones denominados de video digital definen la composición de las muestras de las componentes de imagen cuando tienen la forma *YCrCb*, los estudios llevados a cabo sobre percepción de detalles de imagen han demostrado que el ojo es considerablemente sensible a la luminancia, pero mucho menos a los colores, lo que permite suprimir muestras de las señales de diferencia de color sin pérdida apreciable de calidad para una visión promedio humana; dando lugar a una reducción considerable de la información de video [15].

Los patrones de submuestreo comúnmente utilizados para la crominancia son los siguientes [2]:

- ***Patrón 4:4:4:*** En este formato le corresponden igual número de muestras a cada componente, *YCrCb*, es decir, se tiene la misma relación de píxeles tanto horizontal como verticalmente.

- ***Patrón 4:2:2:*** En este formato existen *dos* muestras de *Y* por *una* muestra de *Cr y Cb* respectivamente en la dirección horizontal, manteniendo la misma relación entre componentes para la dirección vertical.

- ***Patrón 4:1:1:*** En este formato existen *cuatro* muestras de *Y* por *una* muestra de *Cr y Cb* respectivamente en la dirección horizontal, manteniendo la misma relación entre componentes para la dirección vertical.

- ***Patrón 4:2:0:*** En este formato existen *dos* muestras de *Y* por *una* muestra de *CrCb* en las dirección horizontal y vertical, con lo cual se logra disminuir la cantidad de información la Crominancia con respecto a la Luminancia en un factor de 2:1.

El Comité Internacional Consultivo para la Radio (*CCIR*), en su recomendación 601 define el formato de televisión digital estándar (*SDTV*) para el intercambio y transmisión de video digital, como en los estándares analógicos CCIR-601 define dos sistemas el *525/60* y *625/50*, los cuales utilizan el ***patrón 4:2:2***, donde los componentes están cuantizados en ocho bits. En el sistema *525/60* la componente de luminancia de un cuadro tiene dimensiones activas de *720 píxeles x 480 líneas* y las componentes de crominancia tienen dimensiones de *360 píxeles x 480 líneas*.

En el sistema *625/50* los valores correspondientes son 720 píxeles x 576 líneas para luminancia y *360 píxeles x 576 líneas* para crominancia, a pesar de las diferencias entre los dos sistemas se genera la misma cantidad de bits ***165.89 Mbits/s*** [2].

1.3.2.1.- Formatos SIF y QSIF

Para aplicaciones de almacenamiento existe un formato de baja resolución llamado "*formato de entrada fuente*" (SIF), este es un formato progresivo (***4:2:0***), este formato tiene 360 píxeles por línea para la luminancia; debido a que existen dos sistemas para SDTV (CCIR-601) tenemos por lo tanto dos formatos SIF. El primero tiene una componente de luminancia de *360 píxeles x 240 líneas*, las componentes de crominancia tienen *176 píxeles x 120 líneas*, y una razón de cuadros de 30/s.

Mientras que el segundo formato tiene una luminaria de *360 píxeles x 288 líneas*, sus componentes de crominancia tienen una resolución de *176 píxeles x 144 líneas* y una razón de cuadros de 25/s.

Una versión de baja resolución del SIF es el formato cuarto de SIF (QSIF por sus siglas en ingles), tiene la mitad de las dimensiones del SIF en ambas direcciones. De nuevo existen dos versiones, la primera tiene *176 píxeles x 120 líneas* para su luminancia, *88 píxeles x 60 líneas* de crominancia y una razón de cuadros de 30/s; mientras que la segunda tiene una luminancia de *176 píxeles x 144 líneas*, y *88 píxeles x 72 líneas* de crominancia a una razón de 25/s [2].

1.3.2.2.- Formato CIF

El formato CIF fue definido para lograr coincidir con ambos formatos 525/60 y 625/50, en este formato la componente de luminancia tiene una resolución horizontal que es la mitad de ambos sistemas del CCIR-601, una resolución vertical que es la mitad del sistema 625/50 y una resolución temporal que es la mitad del sistema 525/60. La elección intermedia de una resolución vertical de un sistema y la resolución temporal del otro permiten nombrarlo formato intermedio común CIF, este formato CIF es progresivo (***4:2:0)*** y utiliza una razón de cuadros de 30/s [2].

1.3.2.3.- Familia de formatos CIF

Para aplicaciones tales como video sobre redes móviles o video telefonía es posible reducir la razón de cuadros, ésta puede variar entre *15, 10, 12.5, 8.3, 7.5* o *5* por segundo. Existe un número de miembros de mayor y menor resolución que pueden utilizar estas razones (pertenecientes al formato CIF) los principales se muestran en la ***Tabla 1.1***:

	Píxeles por Línea	*Líneas por Cuadro*	*Píxeles por Línea*	*Líneas por Cuadro*
SQCIF	*128*	*96*	*64*	*48*
QCIF	*176*	*144*	*88*	*72*
CIF	*352*	*288*	*176*	*144*
4CIF	*704*	*576*	*352*	*288*
16CIF	*1408*	*1152*	*704*	*576*
	Luminancia		***Crominancia***	

Tabla 1.1.- Familia de formatos CIF.

Todos estos son formatos progresivos y utilizan el ***patrón 4 :2 :0*** [2].

1.3.2.4.- Otros formatos

La *razón de aspecto* simplemente es la valor del cociente que resulta de dividir la resolución vertical entre la resolución horizontal, para las imágenes que forman el video. Alrededor del mundo se emplean diferentes razones dependiendo de la aplicación, por ejemplo la *razón 3:2* es utilizada para

películas de 35 mm; las *razones (1.85 a 1, 2.39:1)* son utilizadas en los formatos de cine, la *razón 16:9* es utilizada para *SDTV* y *HDTV*, a su vez *HDTV* emplea dos tipos de resoluciones *1920 x 1080* y *1280 x 720* pudiendo manejar diferentes tipos de resolución temporal.

Para resoluciones de *1920 x 1080* se emplean las resoluciones temporales de *59.94, 29.97* y *23.97* cuadros/s siendo entrelazado la primera y las otras dos progresivas; en la resolución de *1280 x 720* se utilizan *59.94, 29.97, 23.97* cuadros/s siendo todas ellas progresivas [15].

RESUMEN

En este capitulo se ha expuesto que nuestro campo de interés son las imágenes digitales especialmente aquellas que están formadas por arreglos bidimensionales y en particular las imágenes en movimiento; se describieron aspectos básicos de ellas tales como, su resolución, profundidad de bits, así mismo se hizo mención al hecho de que una secuencia de imágenes digitales forma lo que se conoce como video digital.

Debido a que los primeros sistemas de reproducción de video se desarrollaron basados en tecnología analógica, fue necesario describirlos, puesto que muchos de sus conceptos nos facilitan el entendimiento del video digital; además de esto se expusieron las principales características del video digital, así como sus ventajas y desventajas con respecto al analógico.

Muchos de estos conceptos expuestos nos ayudarán en los siguientes capítulos a comprender adecuadamente el desarrollo de este trabajo de tesis.

CAPITULO II.- TÉCNICAS DE COMPRESIÓN DE VIDEO DIGITAL

El objetivo de este capítulo es dar a conocer los conceptos fundamentales asociados con las técnicas de compresión de video digital, así como exponer las principales ventajas y desventajas de los principales estándares de compresión de video desarrollados, tales como H.261, MPEG-1, MPEG-2 y MJPEG. En este capitulo se verá como las técnicas de compresión de video digital adquieren gran importancia, puesto que permiten un desempeño optimo de los sistemas de transmisión y almacenamiento de video digital.

2.1.- Compresión

Como se vio en la ***sección 1.3*** una de las mayores desventajas del video digital es un su gran ancho de banda, como ejemplo se citó una señal *PAL* que produce alrededor de **257.49 Mbits/s** está razón es demasiado alta, lo que imposibilita sus aplicaciones prácticas.

Esta razón tan elevada es la causa de la creación y aplicación de las técnicas de compresión, las cuales han jugado un papel muy importante en el mundo de las telecomunicaciones y los sistemas multimedia donde el ancho de banda es el factor principal. Esto se debe al hecho de que él espectro electromagnético tiene que ser compartido por muchos servicios tales como, telefonía celular, redes inalámbricas, comunicaciones satelitales, etc, lo que obliga al uso eficiente del mismo; consecuentemente la necesidad de comprimir la información en las radiocomunicaciones digitales se ha vuelto un criterio esencial.

Dentro de estas técnicas se encuentran las de compresión de video digital, que son las principales herramientas para reducir la cantidad de información necesaria para una secuencia de imágenes sin perder la calidad juzgada por los seres humanos [15]. Las técnicas de compresión comúnmente denominadas *algoritmos de compresión*, tienen como objetivo, transformar un flujo de datos en un flujo de palabras código, si la transformación es efectiva las palabras código ocuparán menos bits que los datos originales [10].

Con el transcurso de los años, los *algoritmos de compresión* se han vuelto más complejos, como consecuencia se genera un incremento en el poder de computo, la rapidez de la compresión es determinada por el tipo de datos, el algoritmo de compresión y la velocidad del procesador. Con la disponibilidad de microprocesadores de alto desempeño la compresión de datos puede ejecutarse por medio de software, a veces se necesita mas potencia y para estas aplicaciones se utiliza un chip de alto desempeño como un *DSP* o hardware especializado [18].

2.1.1.- Necesidad de la compresión

En sistemas digitales existen varias razones para usar la compresión de datos [10]:

- Eficientar el espacio de almacenamiento (La compresión permite aumentar el tiempo de reproducción de un dispositivo de almacenamiento)

- Conservación del ancho de banda para la transmisión.
- Reducción del tiempo de transmisión.
- Posibilitar la utilización de Hardware especializado en algún estándar de compresión para aplicaciones de consumo, por ejemplo camcorders digitales y cámaras fotográficas.

La capacidad de almacenamiento y el ancho de banda para servicios de transmisión de datos digitalizados ha crecido y continuará creciendo, las nuevas aplicaciones y servicios particularmente para casa y/o oficina que utilizan voz, audio, imágenes y video digitales requieren incrementos exponenciales sostenidos para el almacenamiento y transmisión, que sin las técnicas de compresión no serían posibles.

Sin embargo, aún la compresión mas efectiva que se realiza en la actualidad, no puede superar todas las demandas de almacenamiento y transmisión de datos, texto, gráficos, voz, audio, imágenes y video. La ***Figura 2.1*** muestra el almacenamiento requerido con y sin compresión para diferentes tipos de datos digitales, lo que nos permite visualizar la importancia de las técnicas de compresión digital (para su comparación se supone una página de texto, una imagen y cinco minutos de grabación de audio y video en diferentes formatos) [10].

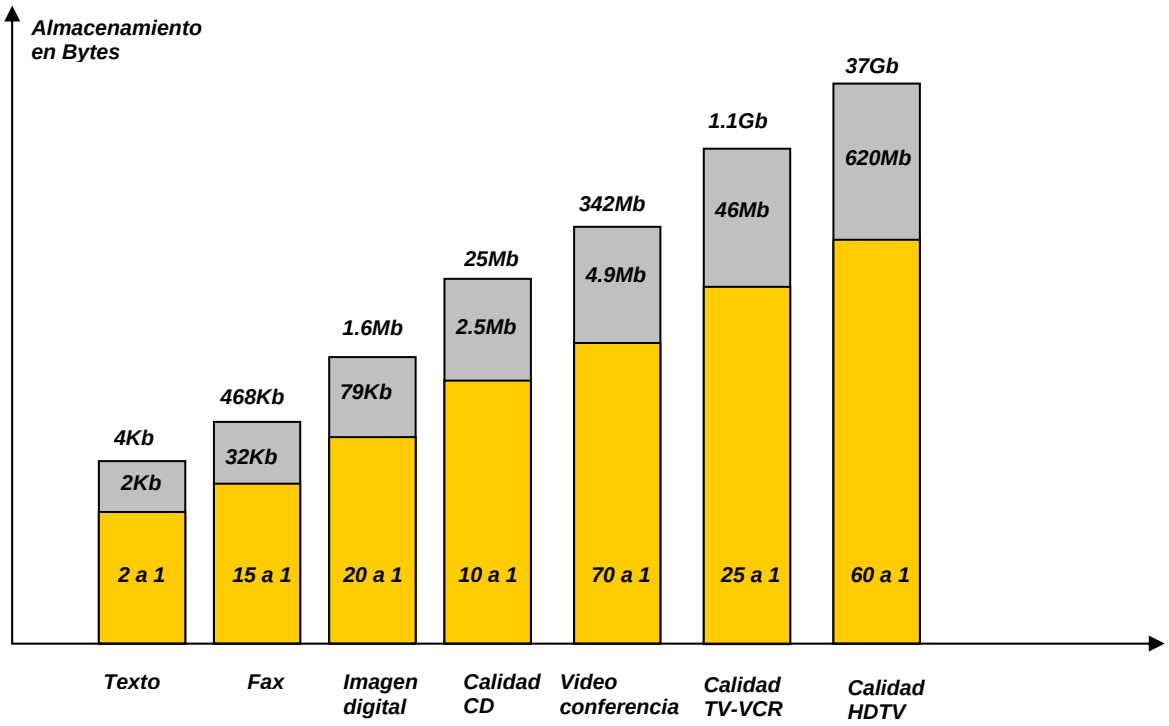


Figura 2.1.- Razones de compresión

2.1.2.- Ventajas de la compresión

La ventaja más grande de la compresión se da en la reducción de datos, esta mejora los costos de transmisión cuando su capacidad es fija, además genera un uso eficiente recursos de almacenamiento. Son muchas y muy variadas las aplicaciones que han tomado ventaja de las técnicas de compresión a continuación se citarán las principales [18]:

Con el advenimiento del *Fax* se incrementó la necesidad para realizar una transmisión rápida de documentos, sobre redes de conmutación telefónicas publicas (*PSTN*), lo cual implicó reducir el número promedio de bits por página; a esto se le llamó compresión digital de documentos (que fue una de las primeras técnicas digitales de compresión implementadas); con la aparición de la videoconferencia la necesidad de la compresión digital se volvió más importante.

Para almacenamiento de video digital sobre un CD, la compresión es absolutamente necesaria, es la única manera de cumplir con los parámetros requeridos por la industria del entretenimiento (por ejemplo mantener el espacio de almacenamiento del CD con una calidad visual aceptable), mientras se conserva el tiempo de duración de las películas, la cual puede exceder las dos horas.

Un canal de TV por cable puede tener entre 4 y 10 programas audio – visuales comprimidos, alternativamente un canal simple de transmisión de 6 MHz puede transportar *HDTV* comprimida generando significativamente mejoras en la calidad de audio e imagen sin ancho de banda adicional.

Las técnicas de compresión posibilitan el servicio de video sobre demanda haciéndolo económicamente factible. En estaciones de trabajo diseñadas para editar audio y video, el material es almacenado en discos duros para su rápido acceso, la compresión hace posible almacenar una gran cantidad de archivos de video digital. La Videotelefonía celular inalámbrica debe operar a razones de pocas docenas de kilobits por segundo, lo cual solo se puede alcanzar a través de una gran compresión de datos.

Todas estas aplicaciones han mejorado de manera significativa su ancho de banda para transmisión, así como su capacidad de almacenamiento a cambio de tener que incrementar el poder de procesamiento.

2.2.- Codificación perceptiva de video

En los sistemas digitales los datos se suelen agrupar en dos categorías [10]:

- Datos simbólicos tales como una letra, una figura, un carácter, una marca o alguna combinación de los mismos, estos representan algo que el ser humano puede reconocer a *priori*.
- Existen también los datos difusos, donde el significado y propiedades de la estructura no han sido extraídos y aun no se convierten ha algo que el ser humano podría reconocer; la voz, el audio, las imágenes y el video cuando se representan por medio digital son ejemplos de datos difusos.

Los *algoritmos de compresión* de datos de tipo simbólico y difuso operan diferentemente, lo que genera que a pesar de que existen muchas técnicas de compresión, solo pueden caer en dos categorías: *compresión de datos sin pérdidas* y *con pérdidas* [17].

Para comprimir datos simbólicos solo se aplica la *compresión sin pérdidas*, debido a que una reproducción exacta del dato original usualmente es requerida por aplicaciones que se utilizan en, negocios, programas de computación, correo electrónico y aplicaciones científicas, las cuales solo permiten la representación exacta de la información.

En estas aplicaciones la pérdida de algún bit, en un carácter de texto, dato numérico o programa de computadora es inaceptable, claramente en programas de computadora la corrupción de un bit puede ser catastrófica, debido a esto la *compresión sin pérdidas* generalmente es restringida a factores de compresión alrededor de 2 : 1 o 3 : 1.

En contraste los datos difusos se pueden comprimir hasta factores de 100 a 1 o más dependiendo del tipo de datos, de la efectividad de los algoritmos de compresión y de la pérdida de información aceptable. Si la compresión de datos es con pérdidas la información no critica es removida y la descompresión de datos no puede ser exactamente igual al dato original, sin embargo el resultado debe ser una aproximación aceptable donde las percepciones sensoriales sean adecuadas, este tipo de compresión es la que se necesita para los datos de tipo difuso [10].

Muchos avances en la compresión de datos difusos han sido posibles por la explotación de los sistemas humanos auditivo y visual; los seres humanos quienes usualmente son los receptores de los datos de tipo difuso no necesitan o no pueden usar toda la información capturada durante la digitalización. Han sido creados modelos poderosos y complejos para datos de video usando lo que se describe como *técnicas de codificación perceptual* que explotan las limitaciones de los ojos humanos.

La compresión de datos de tipo difuso puede involucrar pérdidas desechando bits que no se necesitan para la reproducción de video, puede perderse alguna información que generará diferencias entre los datos sin compresión y los comprimidos, pero los humanos debido a sus límites visuales hacen que estas diferencias sean aceptables para muchas aplicaciones. El análisis estadístico de las señales de video indica que existe una fuerte correlación entre cuadros de imagen sucesivos y con los elementos mismos de la imagen, teóricamente la decorrelación de esas señales disminuye el ancho de banda sin afectar significativamente la resolución de la imagen, la respuesta del sistema visual humano para cierta información *espacio-temporal* puede ser explotada para reducir las razones de bits, mientras se mantiene una calidad aceptable de las imágenes [15].

Una cámara de video que capture 30 cuadros por segundo de una escena estacionaria produce cuadros muy similares uno después del otro, la compresión permite remover la información superflua que está contenida en un cuadro. La señal de video digital contiene una cantidad significativa de *redundancia estadística*, ya que las muestras son muy similares entre sí. Tal *redundancia estadística* puede ser removida sin destruir información útil [18].

La datos originales son reducidos por el compresor; los datos comprimidos son entonces pasados a través de un canal de comunicación o almacenados en algún dispositivo y regresados a su forma original por el expansor, la razón entre la cantidad de los datos originales y los datos comprimidos es llamada ***factor de compresión***, algunas veces un compresor y un expansor en serie son referidos como el "compander", el compresor puede ser igualmente referido como el codec y el expansor como el decoder, estos dos términos también pueden ser llamados *codecs* [17].

Los compresores de video digital son diseñados para eliminar la mayor parte de la *redundancia estadística* sin afectar el contenido de información de la señal; la *redundancia estadística* para una señal de video digital se puede dividir en tres tipos:

- Redundancia espacial
- Redundancia Temporal
- Redundancia intersímbolos

Cada una de ellas se detalla en las siguientes secciones.

2.2.1.- Reducción de la redundancia espacial de video

La redundancia espacial es definida como la "información repetida que se puede descartar", presente en regiones formadas por píxeles que pertenecen a una imagen de la secuencia de video.

El tipo de compresión que permite reducir la redundancia espacial entre píxeles está basada comúnmente en la codificación por transformada y se le denomina *codificación Intra*. En un cuadro de la secuencia de video la *codificación intra* trabaja en dos dimensiones sobre los ejes espaciales horizontal y vertical, un análisis de una imagen típica revela que mientras existe contenido de frecuencias altas debido a los detalles espaciales finos en las áreas de la imagen, existen pequeñas cantidades de energía para tales frecuencias. Sin embargo las imágenes comúnmente contienen áreas considerables que están formadas por píxeles de un valor muy similar entre si, a estos valores les corresponden frecuencias espaciales bajas.

La compresión puede ser obtenida tomando ventaja del hecho que la amplitud de los componentes espaciales disminuye con la frecuencia, también es posible tomar ventaja del hecho que la sensibilidad del ojo en las frecuencias espaciales altas es reducida; si la frecuencia espacial de las imágenes es dividida en bandas de frecuencia, las bandas de mayor frecuencia pueden ser descritas por pocos bits, debido a que sus amplitudes son más pequeñas. Existen ciertas transformaciones matemáticas las cuales permiten describir a las imágenes en el dominio de la frecuencia (***sección 3.1***)[17].

La *codificación intra* es usada para remover las redundancias espaciales en las imágenes mapeando los píxeles hacía un dominio transformado para reducir los datos, la ventaja de la *codificación intra* para la compresión de datos es que la energía contenida en las imágenes en la mayoría de las escenas naturales está principalmente concentrada en una región de baja frecuencia y

por lo tanto se obtienen unos pocos coeficientes transformados, esos coeficientes pueden ser cuantizados descartando los coeficientes de menor amplitud [15].

Los vectores base de la *Transformada Coseno Discreta (DCT)* son la elección ideal para la realización de la *codificación intra*. La razón se debe a que la variación suave de estos vectores produce que la energía de la mayoría de las imágenes en tonos reales se concentre en unos pocos coeficientes (esta transformación se analizará a fondo, así como sus efectos en el ***capitulo 3***).

Debido a que en una secuencia de imágenes, los píxeles están correlacionados en las direcciones vertical, horizontal y temporal de la secuencia de video una elección natural para la transformación es una *DCT* en tres dimensiones, sin embargo cualquier transformación en el dominio temporal requiere de almacenamiento de varios cuadros lo cual produce grandes retardos, que restringen la aplicación de la *codificación intra*, debido a esto la transformación es confinada a dos dimensiones.

2.2.2.- Reducción de la redundancia temporal de video

La redundancia temporal es definida como la "información repetida que se puede descartar", que se presenta en regiones formadas por píxeles que varían con respecto al tiempo o lo que es lo mismo, entre diferentes imágenes de la secuencia de video.

El tipo de compresión que permite reducir la redundancia temporal entre imágenes está basada comúnmente en la codificación de sus diferencias y se le denomina *codificación Inter*. La redundancia temporal es reducida usando la diferencia entre imágenes sucesivas, para las partes estáticas de una secuencia de imágenes la diferencia temporal estará cercana a cero, las partes que cambian entre cuadros son debido a variaciones en la iluminación o al movimiento de los objetos, resultando una imagen de error la cual se necesita para ser codificada, los cambios en la imagen debidos al movimiento pueden ser reducidos significativamente si el movimiento del objeto puede ser estimado y su diferencia es tomada con respecto a alguna imagen de referencia.

Para realizar la *codificación Inter*, primero tiene que ser encontrado el movimiento que está presente en el video, este proceso se conoce como estimación de movimiento. Esta técnica se basa en el algoritmo de igualación de bloque (*BMA* por sus siglas en Inglés), en el cual un cuadro de la secuencia de video es dividido en bloques de *NxN* píxeles.

Estos bloques se deberán igualar con los bloques correspondientes del cuadro anterior de la misma secuencia, para esto se considera que los bloques se desplazan dentro de una ventana cuyas dimensiones son (*N+2W*)*x*(*N+2W*), ***Figura 2.2***, siendo *2W* el número máximo de píxeles que se puede desplazar el bloque en las direcciones horizontal y vertical dentro de la ventana de búsqueda.

Una vez que los píxeles del bloque actual coinciden con los del bloque anterior dentro de la ventana, es posible cuantificar su movimiento, las coordenadas que indican este desplazamiento son conocidas como *vectores de movimiento*.

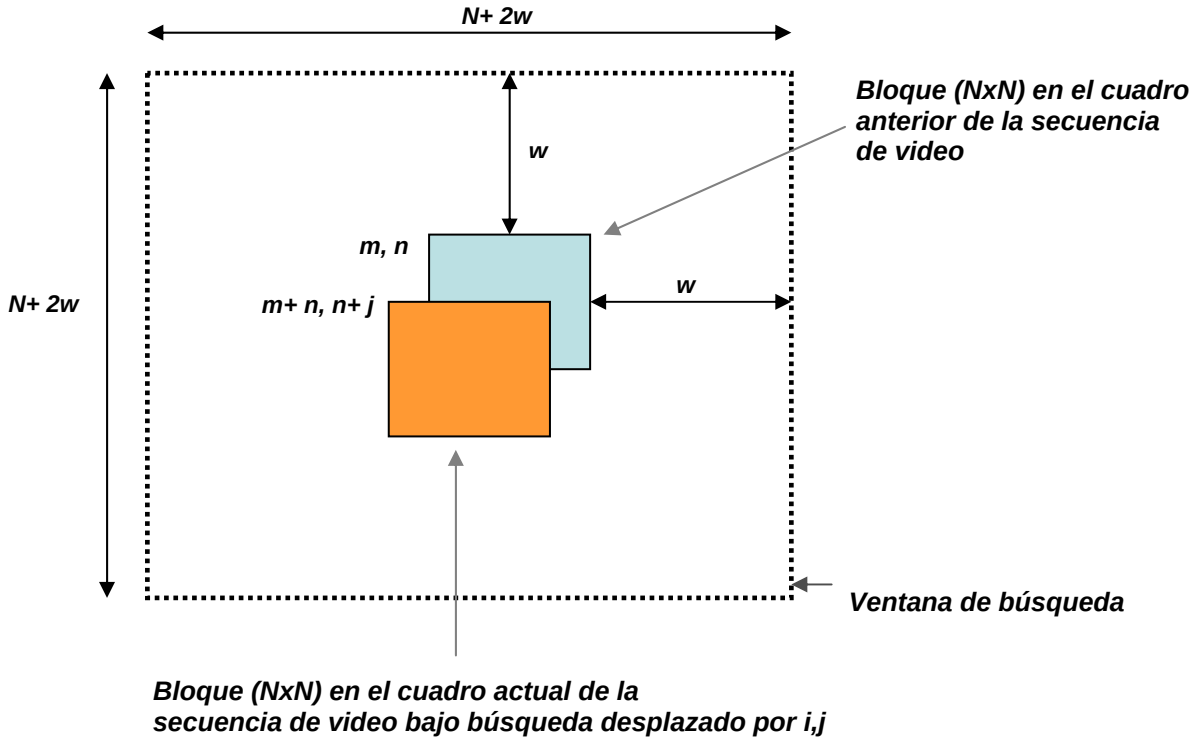


Figura 2.2.- Estimación de movimiento

Una vez que se encuentran los *vectores de movimiento* para todas las áreas de la imagen, la imagen de referencia (cuadro anterior) es desplazada de acuerdo a esos vectores, como resultado se obtiene una *imagen estimada* que es comparada con la imagen actual, para producir un error llamado *residuo*. El *residuo* es transmitido junto con los vectores de movimiento hacia el receptor, una vez que en él se genera la imagen de referencia, está se desplazará de acuerdo a los *vectores de movimiento* transmitidos, para recrear la *imagen estimada* y después el *residuo* es sumado para obtener la imagen original [17].

Reordenamiento de imágenes: Para realizar una codificación altamente eficiente se sugiere que no todas las imágenes de la secuencia de video se codifiquen de la misma manera, debido a esto se identifican tres tipos de imágenes en la secuencia de video, (*imágenes I,P,B*) el conjunto de estos tipos de imágenes forman los denominados grupos de imágenes (*GOP´s*) [15].

- *Imágenes I*: El primer tipo de imágenes es llamado *I*, son codificadas sin la referencia de una imagen previa, estas permiten un punto de acceso para decodificar la secuencia, estás imágenes son llamadas cuadros – intra.
- *Imágenes P*: Corresponden a las cuadros generados por la estimación de movimiento, la diferencia entre la imagen actual y la anterior, así como por la información de los vectores de movimiento.

- *Imágenes B:* Corresponden a una imagen estimada bidireccional, están formadas por información complementaria de cuadros anteriores y posteriores a un cuadro de referencia. En el proceso de formación de *GOP´s* las imágenes *B* se consideran de información complementaria porque contienen lo que falta para obtener cuadros con los datos tomados de las imágenes de tipo *I* o *P*.

Los *GOP´s* permiten acceder aleatoriamente en la secuencia, la primera imagen codificada en el grupo es una imagen de tipo *I*, seguida por un arreglo de imágenes *P* y *B* tal como se muestra en la ***Figura 2.3***.

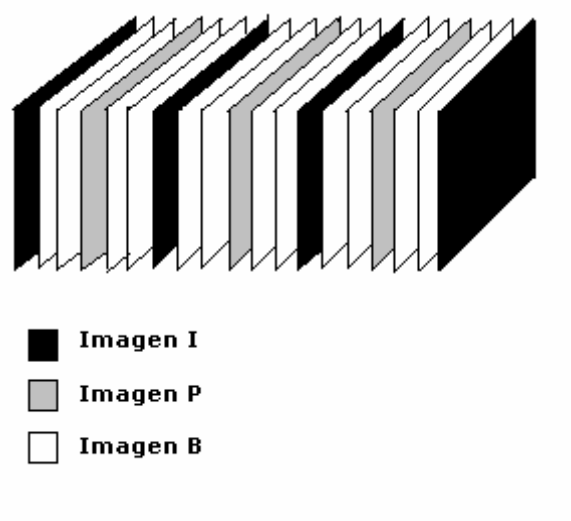


Figura 2.3.- Grupo de imágenes (GOPs)

Esta agrupación tiene sus limitaciones por ejemplo en el caso que suceda un error en la transmisión, cada imagen subsecuente *P* o *B*, este se propagará a lo largo de todo el *GOP*.

2.2.3.- Reducción de la redundancia intersimbolos de video

La redundancia intersímbolos es definida como la "información no útil" que se encuentra presente una vez que se ha eliminado tanto la redundancia espacial, como temporal y corresponde a los códigos resultantes, los cuales deben ser optimizados de acuerdo a su probabilidad de ocurrencia.

La *Información* es definida como un conjunto de características o elementos que proporcionan conocimiento acerca de algún objeto o evento [17]. Una característica de la información es que es impredecible o es un dato de elemento sorpresa; pues de acuerdo a la teoría de *Shannon* cualquier señal que es totalmente predecible no contiene información.

A principios de la década de 1950 la teoría de la información y los conceptos de probabilidad, formaron la base para la creación de una clase de algoritmos para la compresión de datos; los cuales usan patrones de longitud variable para codificar símbolos individuales basados en su frecuencia de ocurrencia [10].

Por ejemplo en un texto en Inglés es posible desarrollar un análisis estadístico de la frecuencia con la cual las letras particulares son usadas, esto es explotado por la *codificación de longitud variable* que asigna códigos cortos a las letras que ocurren frecuentemente, y asigna códigos largos a las letras que no ocurren tan frecuentemente, lo que da como resultado una codificación sin pérdidas.

El primero en utilizar esta idea fue *Samuel Morse*, quién en 1843 desarrolló un código eficiente consistente de puntos, rayas y espacios que permiten transmitir mensajes eléctricamente primero utilizando cables y después ondas electromagnéticas por medio de la radio.

Por ejemplo la letra *"E "* es la más frecuente en Inglés y es representada como un simple punto en el código Morse, una letra muy infrecuente es la Z a la cual se le asigna un patrón complejo; hay que recalcar que este tipo de códigos solo son eficientes en señales que se conocen a "priori" por ejemplo si el código Morse se utiliza en otro lenguaje, su representación se puede tornar significativamente menos eficiente porque las propiedades estadísticas son ampliamente diferentes, por ejemplo la letra Z es muy común en el idioma Checo [17][16].

2.2.3.1.- Codificación Huffman

Existen dos tipos de *codificación de longitud variable* que son empleadas en los estándares de codificación de video, la *codificación Huffman* y la *codificación aritmética*, en este estudio solo se abordará la primera.

La *codificación Huffman* es resultado de estudios acerca de la teoría de la información y probabilidad; la idea general de la codificación estadística es la observación de que tan seguido ocurren los símbolos en particular. Esta técnica fue usada en un principio en el código Morse y fue normalizada por el trabajo de *Shannon, Huffman, Fano* y muchos otros empezando en los últimos años de la década de los 40.

En la *codificación Huffman* el número optimo de bits a usarse para cada símbolo es $|log_2 P|$, donde P es la probabilidad de un símbolo dado, sin embargo debido a que la asignación de palabras código deben realizarse de un número entero de bits, genera que la *codificación Huffman* no sea tan eficiente. Por ejemplo si la probabilidad de los símbolos es 0.33 el número óptimo de bits para codificar el símbolo es alrededor de 1.6 bits pero el esquema de codificación asigna 1 o 2 bits.

También cuando la probabilidad de los símbolos se vuelve muy alta la *codificación Huffman* no es muy óptima, por ejemplo para un símbolo con probabilidad de 0.9 el tamaño del código óptimo debería ser de 0.15 bits, pero la *codificación de Huffman* asigna un valor mínimo de 1 bit de código al símbolo lo cual es seis veces mayor que lo necesario [15].

Los *códigos de Huffman* están construidos por un árbol binario, donde las salidas de este son las probabilidades de los símbolos que deben ser codificados, la manera de generar esté árbol es la siguiente [10]:

1).- Ordenar las probabilidades de menor a mayor

2).- Combinar mediante adicción las dos menores.

3).- Ir al paso 1 hasta que solo queden 2 probabilidades

4).- Retrocediendo en el árbol, generar códigos alternando las asignaciones de 0 y 1.

Como un ejemplo sencillo, se necesita encontrar los *códigos Huffman* del histograma que se muestra en la ***Figura 2.4***.

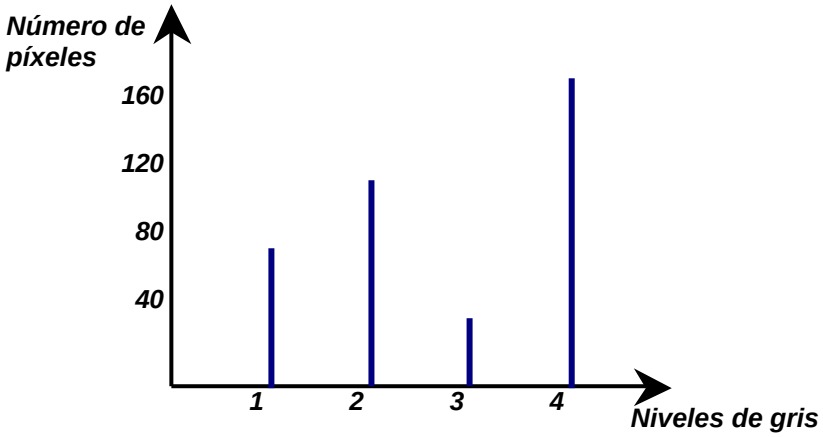


Figura 2.4.- Histograma

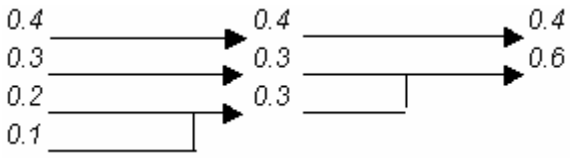
Los valores del histograma se convierten en probabilidades:

$$P_0 = 80/400 = 0.2$$
$$P_1 = 120/400 = 0.3$$
$$P_2 = 40/400 = 0.1$$
$$P_3 = 160/400 = 0.4$$

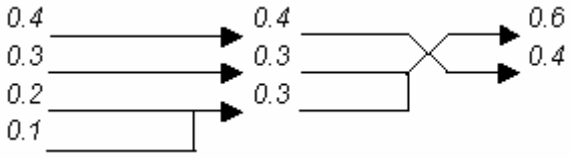
Paso 1*:* Ordenar probabilidades de menor a mayor.

0.4
0.3
0.2
0.1

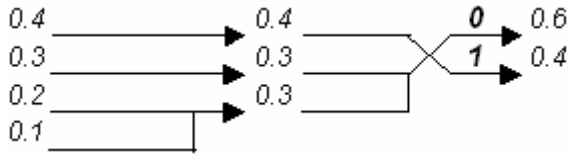
Paso 2: Combinar mediante adición las dos probabilidades menores.



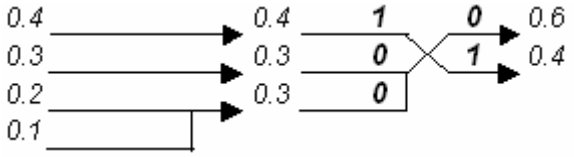
Paso 3: Reordenar de mayor a menor, hasta que solo queden dos probabilidades.



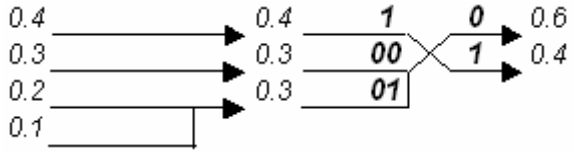
Paso 4: Retroceder en el árbol, generando códigos alternando las asignaciones de 0 y 1.



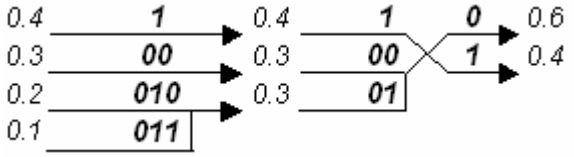
Trasladar los códigos generados hacia sus ramas anteriores de la derecha.



Añadir 0 y 1 en las ramas que se bifurcan en dos.



Repetir el proceso hasta etiquetar la rama principal.



Los *códigos de Huffman* finales para el histograma de la ***Figura 2.4*** se muestran en la ***Tabla 2.1***:

Probabilidad	***Código Huffman***
0.2	***010***
0.3	***00***
0.1	***011***
0.4	***1***

Tabla 2.1.- *Códigos Huffman* finales

En este ejemplo se observa que solo hay dos valores, con tres bits para su representación y un solo valor que se representa con solo un bit, que es justamente el más probable y por lo tanto tiene la menor información en el sentido teórico.

Se debe notar que para un gran número de símbolos, tales como los valores de los coeficientes *DCT*, tal método puede alcanzar grandes cadenas de bits para valores que ocurren raramente volviéndose impráctico; en tal caso normalmente un grupo de símbolos es representado por su sumatoria y combinación de probabilidades, a los cuales se les denomina *Códigos de Huffman modificados*, este método es el que se utiliza en él estándar *MJPEG* (que será descrito con mas detalle en el ***capítulo 3***) [1].

2.3.- Estándares de compresión de video

Con la disponibilidad de la tecnología en los años 80, el Comité Internacional Consultivo de Telegrafía y Telefonía (*CCITT*) estandarizó un codec de vídeo basado en la modulación diferencial por codificación de pulso (DPCM) llamado *H.120,* el codec alcanzó una razón de bits de hasta *2 Mbits/s* para Europa y *1.544 Mbits/s* para Norteamérica, sin embargo, a pesar de tener una buena calidad en su resolución espacial, tuvo muy pobre calidad temporal.

A finales de la década de los 80 fueron propuestos a la Unión Internacional de Telecomunicaciones (*ITU-T* anteriormente *CCITT)* 15 codecs para videoconferencia 14 de ellos estaban basados en la *DCT* y solo uno en la cuantización vectorial (*VQ*), durante el periodo de 1984 – 1988 el grupo *JPEG* se interesó en la compresión de imágenes estáticas, eligieron la *DCT* como la unidad principal de compresión debido a la posibilidad de transmisión de imágenes progresivas, la decisión irrevocable del grupo *JPEG* influencia a la *ITU-T* en favor de la *DCT* sobre la *VQ*. Debido a esto la recomendación de la *ITU-T* para el codec de videoconferencia mostró amplias mejoras en la calidad de la imagen sobre *H.120,* de hecho la calidad de imagen para aplicaciones de videoconferencia se encontró con una calidad razonable a *384 Kbit/s*, esta razón fue extendida a sistemas basados en múltiplos de *64Kbits*, la definición del estándar fue completada a finales de 1989 y es oficialmente conocido como estándar *H.261*.

La estructura de codificación *H.261* está basada en el codec genérico, los principales elementos son el, *estimador de movimiento*, la *codificación por transformada*, la *etapa de cuantización*, el *codificador de longitud variable* y la etapa de control. En algunos codecs la etapa de *estimación de movimiento* puede no ser incluida (en *H.261* es opcional).

La ***Figura 2.5*** muestra un esquema del codificador genérico, en el cual se basan todos los codecs estándares de vídeo, en esta figura se muestra el codec genérico que comprende la eliminación de las redundancias espacial, temporal e intersímbolos (***secciones 2.2.1, 2.2.2*** y ***2.2.3***) [16].

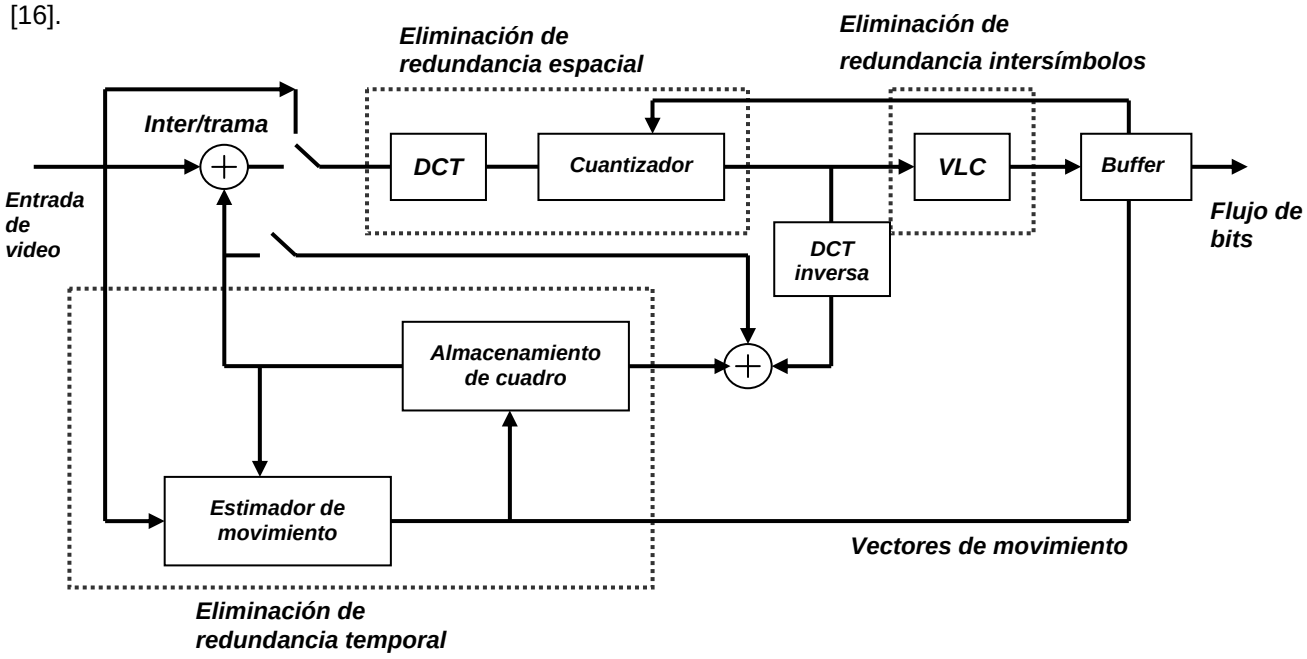


Figura 2.5.- Codificador genérico

Existen distintos codecs de video digital (tales como *H.261, MPEG-1, MPEG-2, MJPEG*), cada uno se diseñó buscando distintos objetivos, un sistema puede ser idóneo para cierto tipo de trabajo pero absolutamente inadecuado para otro y todos ellos tienen sus ventajas y sus incovenientes [15][18][3].

2.3.1.- Estándar H.261

El estándar *H.261* define los métodos de codificación y decodificación de video para una transmisión digital sobre una *Red Digital de Servicios Integrados (ISDN)* con razones de *R x 64* bits/s donde *R* está en el intervalo de *1 – 30*, las razones de video estarán aproximadamente entre *64 kbits/s* y *1920 kbits/s.* Este estándar está diseñado para videotelefonía, videoconferencia y otros servicios audio-visuales [18].

2.3.2.- Estándar MPEG-1

El éxito del estándar *H.261* fue un logro para razones de codificación de video de pocos bits y con calidad razonable, a principios de los 90 el grupo de expertos para imágenes en movimiento (*MPEG*) comenzó a investigar con las técnicas de compresión de video para almacenamiento. El objetivo fue desarrollar un codec de video capaz de codificar eficazmente grandes periodos de tiempo, tales como películas sobre discos duros o CDROM, con un desempeño comparable a los reproductores de video casettes (VHS).

El resultado de está investigación fue la primera generación del grupo *MPEG* la cual es conocida como *MPEG-1* y oficialmente como ISO 11172, es un estándar creado para proveer codificación de video para almacenamiento digital, tal como en un CD o dispositivos ópticos, se diseñó para utilizar la razón de *1.2 Mbits/s* (razón que utiliza el CD), sin embargo en muchas aplicaciones su taza de video está en el intervalo de *1* a *1.5 Mbit/s*, para conseguir estas razones de bits se debe utilizar el formato CIF.

Además de introducir una gran mejora de las herramientas de codificación, las cuales incluyen la sintaxis, la codificación bi-direccional y el control, utiliza también los principios de la codificación espacial que fueron tomados de *JPEG* para imágenes estáticas; a su vez *MPEG-1* especifica compresión de audio hasta para dos canales en sonido estereo y fue optimizado para aplicaciones que utilizan video no-entrelazado a 25 o 30 cuadros/s.

Consecuentemente el formato *MPEG-1* puede proveer calidad tan buena como la provista por las videograbadoras, a su vez el decodificador/reproductor es capaz de proveer las mismas funciones que una VCR convencional, tales como imagen congelada, movimiento hacia delante, movimiento hacia atrás, así como un acceso aleatorio.

Todas estas innovaciones hicieron posible que los decodificadores/reproductores de *MPEG-1* estuvieran disponibles rápidamente en los sistemas multimedia, después de eso fueron incluidos por medio de Hardware, con lo cual muy pronto estuvieron presentes para casi todos los sistemas operativos, así como las plataformas de PC y MAC; otro resultado interesante que también produjo *MPEG-1* fue la creación del video CD (*VCD*) que fue bien aceptado en el mercado.

El estándar *MPEG-1* llegó a ser muy popular, debido a que su algoritmo es muy eficaz (aunque tiene la desventaja de que asimétrico es decir su codificador es mas complejo que su decodificador), gracias a que no solo comprime imágenes estáticas sino que además compara los cuadros presentes con los futuros y los anteriores con lo que almacena las partes del video que cambian y de esta manera se realiza la *estimación del movimiento.*

El principal incoveniente es que debido a su complejidad no es posible sacarle el máximo provecho en tiempo real, ya que es imprescindible apoyarse en hardware especifico para compresión y descompresión y no es recomendable en ningún caso si lo que se desea es edición de video debido a esa misma complejidad [15][18].

2.3.3.- Estándar MPEG-2

Después del desarrollo del estándar *MPEG-1,* se incrementó la necesidad de comprimir video digital para TV, la cual puede ser transmitida vía satélite, así como por vía aérea, Televisión por cable (*CATV*) o realizar simplemente su almacenamiento. Las primeras versiones de *MPEG-1* para video entrelazado, las cuales fueron usadas para la transmisión se llamaron *MPEG-1*$^{+}$, los transmisores de TV que inicialmente rechazaron usar cualquier tipo de compresión pronto adoptaron la nueva generación de *MPEG*, que es llamada *MPEG-2* (y oficialmente es conocida como ISO 13818) para codificación de video entrelazado que maneja razones de bit desde *4* hasta *9 Mbits/s*.

El estándar subsecuente *MPEG-2* fue considerablemente más amplio y más eficaz, por ejemplo *MPEG-2* soporta video entrelazado y *HDTV,* mientras que *MPEG-1* no. El estándar *MPEG-2* se ha vuelto muy importante debido a que ha sido elegido como el esquema de compresión para *Transmisión Digital Terrestre* (*DVB*) y para el *Disco versátil Digital* (*DVD*), le fue añadida la capacidad de soportar video entrelazado, así mismo fue ampliado el intervalo de imágenes (soporta casi cualquier tipo de resolución) así como sus razones de bits, además los decodificadores de *MPEG-2* pueden manejar datos codificados en *MPEG-1*.

MPEG-2 tiene la capacidad de ajustar el número de imágenes estimadas bidireccionales por el decodificador, está técnica puede ser usada para sistemas de telecomunicaciones por ejemplo en aplicaciones tales como video sobre demanda, el cliente puede elegir recibir video de diferente nivel de calidad, en aplicaciones de redes durante la congestión las partes menos importantes pueden ser descartadas.

Debido a las muchas aplicaciones de *MPEG-2*, el estándar es subdividido en *perfiles* y *niveles*; un perfil describe el grado de complejidad mientras que un *nivel* describe el tamaño de la imagen que puede soportar, no todos los niveles son soportados por todos los perfiles. Por ejemplo el perfil más simple de *MPEG-2* no soporta la codificación bi-direccional y solo maneja imágenes *P* e *I,* esto reduce los retardos en la codificación y decodificación y permite utilizar Hardware más simple, el *perfil* principal es diseñado para una gran variedad de aplicaciones como las que ya han sido citadas.

Debido a que la compresión es elevada, cada bit puede afectar a un gran número de muestras de una señal de audio o video, por lo tanto un error en la transmisión o en el almacenamiento de los bits comprimidos puede generar grandes efectos en una región de la imagen o extenderse sobre un gran periodo de tiempo. Así como en *MPEG-1* su desventaja más grande se da en la complejidad de su algoritmo, que es su aspecto más crítico está cuestión es controlada por la evolución de la tecnología que provee dispositivos específicos para su implementación.

En noviembre de 1998 inició la *DVB* con programación de la Corporación Británica de Difusión (*BBC*) de Londres en formato digital codificado en *MPEG-2* y casi al mismo tiempo fue lanzado el satélite *SKY Digital* el cual transmite audio y video codificado en *MPEG-2* directo hacia los hogares [15][18].

Aproximadamente en el año 2014 la existencia de los transmisores NTSC analógicos de video cesará en E. U y solo se transmitirá HDTV con compresión MPEG-2 vía terrestre [15].

2.3.4.- Estándar MPEG-4

El estándar *MPEG-4* usa herramientas de codificación con complejidad adicional para alcanzar factores de compresión mas altos que *MPEG-2*, es una compresión más eficiente de video utilizado en aplicaciones de gráficos por computadora; describe formas, texturas, superficies tridimensionales, además tiene como expectativas el convertirse en parte importante para los desarrollos de Internet y aplicaciones inalámbricas tales como transmisión móvil audio-visual, videoteléfonos, e-mail, bases de datos multimedia, además incluye facilidades para la interactividad, y tiene el apoyo de aplicaciones basadas en la realidad virtual [18].

2.3.5.- Estándar MJPEG

MJPEG significa *JPEG* en movimiento, básicamente consiste en tratar al video como una secuencia de imágenes independientes, realizando la compresión y descompresión de cada una mediante el algoritmo *JPEG*, para luego reconstruir la señal de video, provee una inversión en Hardware más económica para su desarrollo, pudiendo manejar sin ningún problema sistemas basados en *NTSC, PAL,* o *SECAM*. Las principales ventajas de *MJPEG* son que la compresión *JPEG* es muy barata de hacer con Hardware y que soporta casi cualquier tamaño del video que se desea transmitir, esto significa que es posible utilizar imágenes en tamaño *sub-QCIF* hasta los de tamaño completo en *HDTV*.

El hecho de que esté estándar no consiga los índices de compresión alcanzados por los estándares *MPEG-1,2,4*, exige que se trabaje con sistemas de comunicación de transferencia sostenida. Debido a que la arquitectura *MJPEG* no agrupa los cuadros de la secuencia en grupos, implica que los errores de transmisión o pérdida de paquetes solo impactan a una imagen sin propagarse a los demás cuadros.

Las principales desventajas de *MJPEG* son la carencia de productos que lo usen, además de que no eficienta tanto el ancho de banda para su transmisión, en el ***capitulo tres*** se detallará con cuidado este estándar [18].

2.3.6.- Estándar MJPEG 2000

Es un estándar de compresión de imágenes basado en la Transformada Discreta Wavelet (*DWT*), fue creada por el comité *JPEG* que anteriormente creo el algoritmo de *MJPEG*, su objetivo fue mejorar la implementación basada en *DCT*. *MJPEG 2000* puede trabajar con niveles de compresión mayores a los de *MJPEG* sin incurrir en los defectos que este estándar produce, tales como generación de bloques uniformes y aspecto borroso; sus principales desventajas están en que tiende a desenfocar mas la imagen que *MJPEG* y que elimina algunos detalles espaciales y texturas que *MJPEG* si llega a representar [18].

RESUMEN

El objetivo de este segundo capítulo fue el describir que es la compresión digital de datos, se hizo énfasis en la compresión digital de video pues en ella consiste este trabajo de tesis; se expuso que existen dos tipos de compresión, la "compresión sin pérdidas" y la "compresión con pérdidas".

La compresión de video se basa en la compresión con pérdidas, explotando el hecho de las limitaciones de los sistemas audible y visuales humanos, esta técnica se conoce como codificación perceptiva, y elimina la redundancia ya sea espacial, temporal o intersímbolos que está presente en una secuencia de video.

Así mismo se dieron a conocer los principales estándares que se utilizan para la codificación de video, se citaron sus principales aplicaciones, ventajas y desventajas; de lo cual se puede concluir que aquellos que proveen una compresión mas eficiente y se destinan a las aplicaciones más importantes, son los estándares MPEG-2 y MPEG-4, sin embargo sus algoritmo son sumamente complejos y aún no son de dominio general; por otro lado el estándar MJPEG provee una compresión moderada, pero no es tan exigente en los cálculos computacionales para su implementación, pues se basa casi por completo en el estándar JPEG; debido a esto, en este trabajo se ha optado por su implementación en el DSP 6711 de TI.

CAPITULO III.- EL ESTÁNDAR DE COMPRESIÓN PARA VÍDEO MJPEG

Este capítulo tiene como objetivo el describir el estándar de compresión para video digital MJPEG, este estándar provee la metodología utilizada para la realización de este trabajo de tesis. El capitulo está dividido en tres secciones, dos de las cuales hacen una referencia detallada del estándar, la primera sección 3.1 describe de manera amplia la utilización de las Transformada Coseno Discreta Directa (FDCT) y la Transformada Coseno Discreta Inversa (IDCT), estas transformaciones son una de las partes más importantes del sistema compresión – descompresión MJPEG, es por eso que deben ser presentadas de la manera más optima. Las siguientes dos secciones 3.2 y 3.3 permiten conocer y describir el funcionamiento de MJPEG, por último se realiza un ejemplo de codificación y decodificación, lo que permite comprender los conceptos citados a lo largo de este tercer capitulo.

3.1.- Transformadas

Las transformaciones matemáticas son operaciones cuyo fin es facilitar los procedimientos necesarios para resolver los problemas matemáticos, en particular las transformadas integrales son usadas principalmente para la reducción de la complejidad en las ecuaciones matemáticas. El objetivo de las transformaciones matemáticas es trasladar una señal de su espacio original a otro espacio matemático, donde el manejo e interpretación de la información se torne más fácil [4].

En años recientes ha habido un crecimiento respecto a las aplicaciones de las transformadas matemáticas en el *Procesamiento Digital de Señales e imágenes*; esto se debe principalmente al impacto de las computadoras digitales de alta velocidad, seguido del desarrollo de *DSP´s*, y al desarrollo de algoritmos rápidos que tienen como consecuencia, una reducción en los costos computacionales y de memoria en la aplicación de las transformadas. Algunas de las aplicaciones de estas transformadas son las siguientes: procesamiento de imágenes, procesamiento de voz, reconocimiento de patrones, reconocimiento de caracteres, análisis y diseño de sistemas de comunicación, filtrado digital, análisis espectral, compresión de datos, procesos de convolución y correlación, análisis estadístico, además de aplicarse a señales tan variadas como: señales sísmicas, sonar, radar, fuentes biológicas y biomédicas, astronomía, ondas oceanográficas, imágenes satelitales, vibraciones estructurales, rayos x, entre otras [5].

Para cubrir este gran intervalo de aplicaciones es necesario contar con diversas transformadas, las cuales proporcionan resultados diferentes al momento de su aplicación, los criterios principales para utilizar una transformación son: el tipo de señal que se tiene y lo que se desea obtener de ella, por ejemplo la Transformada de Fourier (*TF*) permite encontrar el espectro (los componentes frecuenciales que contribuyen a la formación de esa señal en particular) de una señal dada.

En vista que nuestro campo de interés son las imágenes en movimiento este estudio se centra en la utilización de las transformadas adecuadas para su procesamiento. Como se vio en la ***sección 2.2.1***, uno de los objetivos de la compresión de video es la eliminación de la redundancia espacial, para la realización de esto se propuso utilizar una transformación matemática.

La transformada óptima en el sentido estadístico para el desarrollo de la eliminación de la redundancia espacial, es la Transformada Karhunen-Loeve (*KLT*), está decorrelaciona los coeficientes transformados empaquetando la mayor parte de la energía en unos pocos coeficientes, además de minimizar el error cuadrático medio entre las señales original y reconstruida.

Sin embargo la implementación de la *KLT* involucra la determinación de eigenvalores y sus correspondientes eigenvectores y no existe un algoritmo general para su computación rápida, la *KLT* tiene una propiedad teórica óptima de decorrelación, pero pocas veces es usada en la práctica [12]. Esto se debe principalmente a los requerimientos computacionales adicionales para estimar y resolver el eigensistema, para extraer los componentes principales, además el computo de la transformada directa e inversa es considerado lento requiriendo operaciones del orden de N^2 por bloque de *N* píxeles, finalmente mientras la transformada puede ser óptima en sus bases teóricas el criterio de la distorsión puede no corresponder bien con la percepción visual, por ejemplo el efecto de distorsión de bloque es muy visible a razones altas de compresión, una transformada *KLT* es teóricamente posible pero solo es práctica para conjuntos muy pequeños de imágenes.

En vista de los problemas asociados con la implementación de la *KLT*, han sido utilizadas otras transformadas discretas en el procesamiento de imágenes y señales, de hecho algunas transformadas tales como la *DCT* y la Transformada Discreta de Fourier (*DFT*) son asintóticamente equivalentes a la *KLT* [5]. La *DCT* fue aplicada en el trabajo pionero de Ahmed, Natarajan y Rao, en la cual mostraron que esta transformada en particular es muy cercana a la *KLT* la cual produce coeficientes decorrelacionados [1].

Por lo tanto la *DCT* es una transformación eficiente que descompone una señal en un grupo de señales cosenoidales ortogonales llamadas funciones bases, la transformada toma un conjunto de muestras y nos entrega un conjunto de coeficientes, el número de elementos en ambos conjuntos es el mismo [4].

3.1.1.- Ventajas de la DCT sobre la KLT

Como se mencionó en la ***sección 3.1***, existe una equivalencia asintótica entre la *DCT* y la *KLT*, para longitudes de datos finitos se pueden generar diferentes aproximaciones para la *KLT*. En general existen varias características que son deseables en la *DCT* cuando es usada con el propósito de comprimir datos en sustitución de la *KLT* [12]:

- *Decorrelación de datos:* La transformada ideal decorrelaciona completamente los datos en una secuencia, empaqueta la mayoría de la energía en pocos coeficientes, de esta manera muchos coeficientes tendrán valores pequeños.
- *Funciones base independientes*: Debido a las grandes variaciones estadísticas en los datos, la transformada óptima depende de los datos, esto es particularmente un problema si los bloques de datos no están muy relacionados, es decir, necesita el uso de más de un conjunto de funciones base para alcanzar una alta decorrelación, por lo tanto es deseable un desarrollo óptimo para las transformadas cuyas funciones base son independientes.
- *Implementación rápida:* El desarrollo de algoritmos matemáticos que permiten la implementación de la *DCT* e *IDCT* con una cantidad menor de operaciones, con lo cual se logra optimizar su desarrollo.

3.1.2.- Clasificación de las transformadas

La Transformada Coseno y la Transformada Seno Discreta (*DST*) son miembros de una familia de transformadas unitarias senoidales, que encuentran aplicaciones en procesamiento digital de señales e imágenes y particularmente en sistemas de codificación por transformada para compresión y descompresión de datos. Existen varias versiones de la *DCT* (las cuales se verán mas adelante) los tipos *II* y *III* han recibido mucha atención en *PDS*.

A pesar de ser real, ortogonal y separable sus propiedades son relevantes para compresión de datos y sus algoritmos rápidos han proporcionado un valor computacional práctico. Recientemente la *DCT* ha sido empleada como la herramienta principal de procesamiento para compresión y descompresión de datos en los estándares internacionales de codificación de imágenes y video. Una transformada alternativa usada en sistemas de codificación por transformada es la *DST*, de hecho el uso alternado de las formas modificadas de la *DCT* y la *DST* han sido adoptado en el estándar internacional de codificación de audio en *MPEG-1* y *MPEG-2* [12].

La ***Figura 3.1*** clasifica a las transformadas discretas dependiendo, si son óptimas o sub-óptimas [5]:

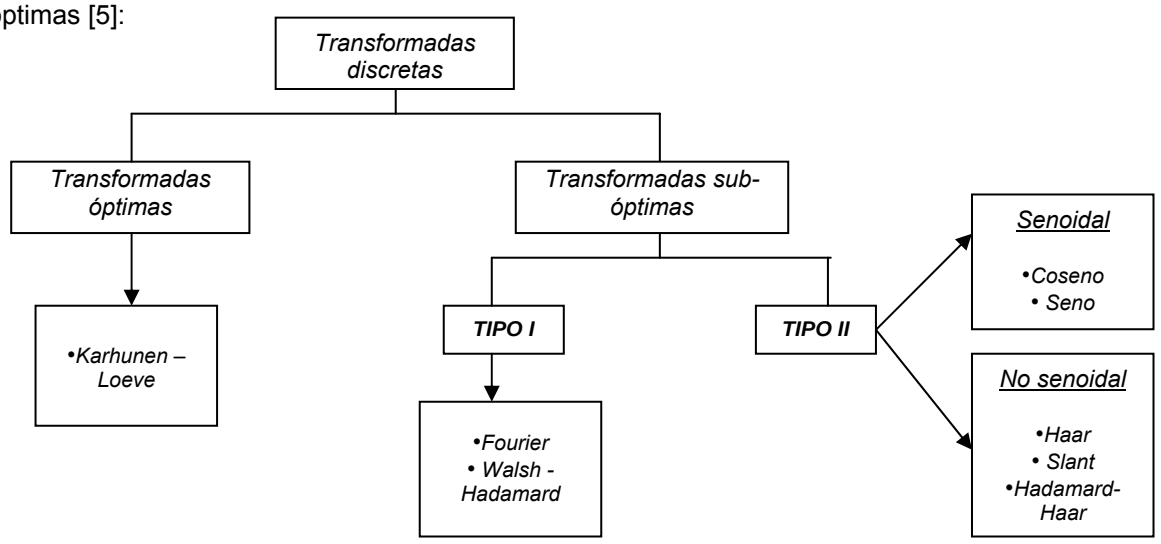


Figura 3.1.- Clasificación de las transformadas

DCT´s y *DST´s* son miembros de una clase de transformaciones unitarias senosoidales desarrolladas por Jain, una transformada unitaria sinusoidal es una transformada lineal invertible cuyo Kernel describe un conjunto completo de cosenos y senos ortogonales discretos (funciones base), la *KLT*, la *DFT*, son miembros de esta clase de transformadas unitarias.

3.1.3.- Transformada Integral Coseno

La Transformada Coseno al igual que muchas otras Transformadas (Laplace, Fourier, Z) son definidas de acuerdo a una operación de Integración la cual muchas veces se denomina *Kernel* de transformación, gracias a esta operación se logra trasladar una señal en su espacio original al espacio de transformación definido por la ecuación **(3.1)**.

Por ejemplo dada una función ***x(t)*** para $-\infty < t < \infty$ su Transformada de Fourier ***X(ω)*** está dada por la ecuación **(3.1)** [4][5]:

$$X(\omega) \equiv \mathbf{F}[x(t)] = \sqrt{\frac{1}{2\pi}} \int_{-\infty}^{\infty} x(t)\, e^{-j\omega t}\, dt \qquad \textbf{(3.1)}$$

Donde

$j = (-1)^{1/2}$

$\omega = 2\pi f$, es la frecuencia angular en radianes.

f , es la frecuencia en Hertz.

La función ***x(t)*** puede ser recuperada por la Transformada Inversa de Fourier (*IFT*) que se define en la ecuación como:

$$x(t) \equiv \mathbf{F}^{-1}[X(\omega)] = \sqrt{\frac{1}{2\pi}} \int_{-\infty}^{\infty} X(\omega)\, e^{-j\omega t}\, d\omega \qquad \textbf{(3.2)}$$

Si ahora la función ***x(t)*** está definida para $0 \leq t < \infty$ su Transformada Coseno de Fourier (*FCT*), $X_c(\omega)$ está definida por la ecuación:

$$X_c(\omega) \equiv \mathbf{F_c}[x(t)] = \sqrt{\frac{2}{\pi}} \int_{0}^{\infty} x(t)\, \cos(\omega t)\, dt \qquad \textbf{(3.3)}$$

Así mismo la Transformada Inversa Coseno de Fourier (*IFCT*), se define por la siguiente ecuación:

$$x(t) \equiv \mathbf{F_c}^{-1}[X_c(\omega)] = \sqrt{\frac{2}{\pi}} \int_{0}^{\infty} X_c(\omega)\, \cos(\omega t)\, d\omega \qquad \textbf{(3.4)}$$

Las ecuaciones **(3.3)** y **(3.4)** definen la Transformada Coseno en su forma continua de la misma manera que las ecuaciones **(3.1)** y **(3.2)** definen a la Transformada de Fourier.

3.1.4.- Transformada Coseno discreta unidimensional

Debido a las definiciones del apartado anterior se observa que el kernel de la transformada Coseno de Fourier está dado por la ecuación **(3.5)** [5]:

$$K_c(\omega,t) = \cos \omega t \qquad (3.5)$$

Si se considera $\omega_n = 2\pi m\, \delta f$ y $t_n = n\, \delta t$, se está muestreando la frecuencia, angular y el tiempo respectivamente, donde ***m*** y ***n*** son enteros, la ecuación anterior se puede rescribir en la ecuación **(3.6)** como:

$$K_c(\omega_n,t_n) = K_c(2\pi m\, \delta f, n\, \delta t) = \cos(2\pi mn\, \delta f\, \delta t) \qquad (3.6)$$

Por definición se considera que $\delta f\, \delta t = 1/2N$, donde ***N*** es un número entero, por lo tanto se tiene que la ecuación **(3.6)** se transforma ahora en la ecuación **(3.7)**:

$$K_c(m, n) = \cos((mn\pi)/N) \qquad (3.7)$$

La ecuación (3.7) representa el *kernel* discretizado de la Transformada Coseno. De acuerdo a este *kernel* se definen cuatro formas para la Transformada Coseno Discreta, las cuales se pueden observar a partir de las ecuaciones **(3.8)**, **(3.9)**, **(3.10)** y **(3.11)**, estas ecuaciones algunas veces son denominadas *DCT´s* tipos (*I, II, III* y *IV*).

DCT-I

$$[Cn]_{mn} = (2/N)^{1/2} (K_m K_n \cos((mn\pi)/N)) \qquad m,n = 0,1,2,....N \quad (3.8)$$

DCT-II

$$[Cn]_{mn} = (2/N)^{1/2} (K_m \cos((m(n+0.5)\pi)/N)) \qquad m,n = 0,1,2,....N \quad (3.9)$$

DCT-III

$$[Cn]_{mn} = (2/N)^{1/2} (K_n \cos((n(m+0.5)\pi)/N)) \qquad m,n = 0,1,2,....N \quad (3.10)$$

DCT-IV

$$[Cn]_{mn} = (2/N)^{1/2} (\cos(((n+0.5)(m+0.5)\pi)/N)) \qquad m,n = 0,1,2,....N \quad (3.11)$$

Donde

$$Kn = \begin{cases} 1 & \text{Si } n \neq 0 \\ \frac{1}{\sqrt{2}} & \text{Si } n = 0 \end{cases} \quad y \quad Km = \begin{cases} 1 & \text{Si } n \neq 0 \\ \frac{1}{\sqrt{2}} & \text{Si } n = 0 \end{cases}$$

$[C_n]_{mn}$ es el elemento mn-ésimo de la matriz *DCT*.

Las formas *DCT-II* y *DCT-III* proveen una excelente decorrelación estadística y son la mejor aproximación para la optimización de la *KLT* tomando en consideración, un número de criterios de desempeño (eficiencia en el empaquetamiento de energía, razón de distorsión). La importancia de las *DCT-II* y *DCT-III* está acentuada por la existencia de algoritmos "rápidos" para su implementación [12].

Con base en la definiciones *DCT-II* y *DCT-III*, la Transformada Discreta Coseno Directa (*FDCT*) unidimensional se define para un conjunto de ocho muestras en la ecuación [1]:

$$F(u) = \frac{C(u)}{2} \sum_{x=0}^{7} f(x) \cos\left[\frac{(2x+1)u\pi}{16}\right] \qquad \textbf{(3.12)}$$

$$u \rightarrow 0:7$$

La definición matemática de la Transformada Inversa Coseno Discreta (*IDCT*), para un conjunto de ocho muestras se define en la ecuación:

$$f(x) = \sum_{u=0}^{7} \frac{C(u)}{2} F(u) \cos\left[\frac{(2x+1)u\pi}{16}\right] \qquad \textbf{(3.13)}$$

$$x \rightarrow 0:7$$

Donde

$$C(u) = \begin{cases} 1 & \text{Si } u > 0 \\ \frac{1}{\sqrt{2}} & \text{Si } u = 0 \end{cases}$$

F(u) es el valor de un coeficiente DCT unidimensional.

f(x) es el valor de una muestra unidimensional.

Como ya se ha señalado la *DCT* es una transformación que descompone una señal en un grupo de señales senoidales ortogonales llamadas funciones base, estas funciones base de acuerdo a la definición de la ecuación **(3.12)** son generadas por el término Coseno y a su vez son escaladas por el factor ***C(u)***, estas funciones junto con los valores originales de las muestras son los encargados de generar los coeficientes *DCT*.

En la ***Figura 3.2*** se muestran las funciones base para un conjunto de ocho muestras, el eje de las ordenadas contiene la amplitud de las señales base sin escalar y el eje de las abscisas, al número correspondiente de cada muestra [6].

En la ***Figura 3.2*** se pueden ver un conjunto de ocho formas de onda de diferente amplitud, cada una formada por ocho elementos; la primera corresponde al coeficiente *DCT* cero y es simplemente una constante, mientras que las otras siete muestran un comportamiento alternado conforme la frecuencia se incrementa; el coeficiente que corresponde a la función base constante es llamado coeficiente de *DC*, mientras que las otras funciones base corresponden a los denominados coeficientes de *AC* [1].

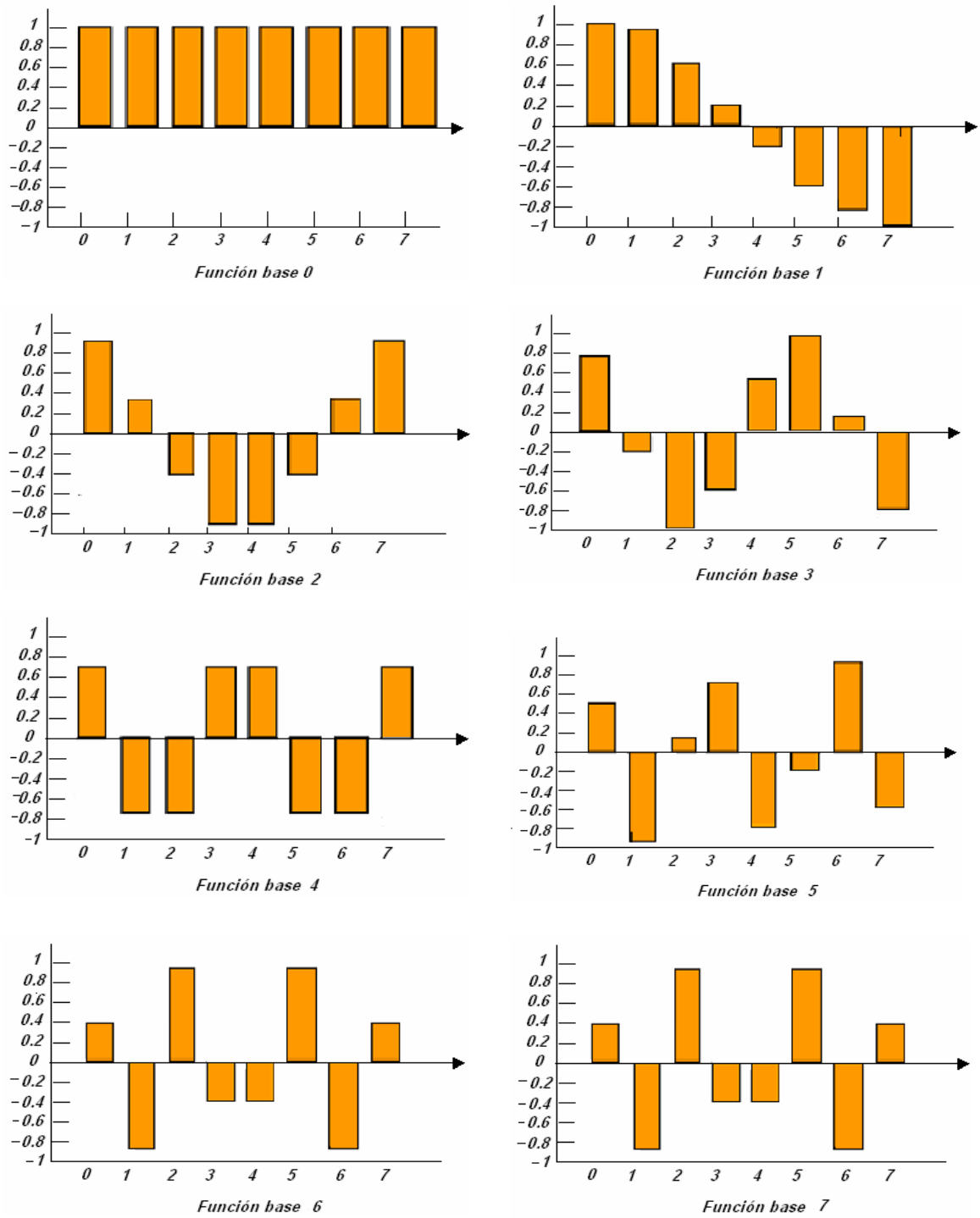


Figura 3.2 .- Funciones base de la DCT unidimensional

Los valores de la primera función son obtenidos de la siguiente manera: por corresponder al primer coeficiente *DCT* el valor de ***u*** es igual cero, debido a esto el termino Coseno proporciona un valor de 1, no importando la variación de x, después de eso se debe escalar por el factor ***C(u)*** al cual le corresponde el valor de *0.7071*, que debe ser dividido entre *2*, lo que da como resultado *0.3536*, para todas las muestras de esta función base. Este procedimiento es el utilizado en la generación de todas las funciones base de la ***Figura 3.2***.

Por ejemplo considérese el vector

$$\boldsymbol{a} = [\ 84\ 84\ \ 90\ \ 92\ \ 88\ \ 66\ \ 77\ \ 86\ ]$$

Que representa el conjunto de ocho muestras que se desea transformar por la aplicación de la *DCT-unidimensional*, la obtención del vector transformado $\boldsymbol{a_T}$, se puede lograr por la aplicación de la ecuación **(3.12)** o bien si se conocen a priori los valores de las funciones base escaladas por el factor **C(u)**, en cuyo caso solo es necesario realizar una suma de productos.

Para obtener el coeficiente de *DC* se realizan las siguientes operaciones

(84x.3536) + (84x.3536) + (90x.3536) + (92x.3536) + (88x.3536) + (66x.3536) + (77x.3536) + (86x.3536) =235.8512

Este resultado es el valor del coeficiente *DC*, de igual manera se sigue el mismo procedimiento en la obtención del primer coeficiente de AC, solo que ahora se utilizan los valores de la función base 1.

(84x.4904) + (84x.4157) + (90x.2778) + (92x.0975) + (88x-.0975) + (66x.2778) + (77x-.4157) + (86x-.4904) = 8.9863

Utilizando el mismo procedimiento, se encuentran todos los componentes del vector de coeficientes

$$\boldsymbol{a_T} = [\ 235.8512,\ \ 8.9863,\ \ -3.6627,\ \ -14.3948,\ 11.6673,\ \ \ 0.0157,\ \ -4.2231,\ \ 5.8765\ ]$$

Al aplicarle la transformada *IDCT* al vector $\boldsymbol{a_T}$ se obtienen nuevamente los valores originales contenidos en el vector ***a***. En el vector $\boldsymbol{a_T}$ es posible apreciar como la mayor parte de la energía del vector ***a*** se concentra en el coeficiente de *DC*, con respecto a sus otros elementos, que presentan valores mucho mas bajos.

El número total de operaciones que la realizan la *DCT* e *IDCT unidimensionales* para un conjunto de ocho elementos, siguiendo el criterio de este ejemplo es *120* (*56 sumas y 64 multiplicaciones*).

3.1.5.- Transformada Coseno Discreta bidimensional

La *DCT* para dos dimensiones se puede generar a partir del productos de dos *DCT unidimensionales*. Con base en este criterio, la definición matemática de la *FDCT* en dos dimensiones para un conjunto de 64 elementos, agrupados en una matriz de *8x8*, se muestra en la ecuación **(3.14)** [4]:

$$F(u,v) = \frac{C(u)}{2}\frac{C(v)}{2}\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right] \qquad \textbf{(3.14)}$$

$$U, V \rightarrow 0:7$$

A su vez la definición matemática de la *IDCT* en dos dimensiones para un conjunto de 64 coeficientes agrupados en una matriz de *8x8*, se muestra en la ecuación **(3.15)**:

$$f(x,y) = \sum_{u=0}^{7}\sum_{v=0}^{7}\frac{C(u)}{2}\frac{C(v)}{2}F(u,v)\cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right] \qquad \textbf{(3.15)}$$

$$x, y \rightarrow 0:7$$

Donde

$$C(v) = \begin{cases} 1 & \text{Si } v > 0 \\ \frac{1}{\sqrt{2}} & \text{Si } v = 0 \end{cases}$$

$$C(u) = \begin{cases} 1 & \text{Si } u > 0 \\ \frac{1}{\sqrt{2}} & \text{Si } u = 0 \end{cases}$$

f(x,y) Es el valor de una muestra bidimensional, para los índices ***x***, ***y*** de la matriz de elementos.

F(u,v) Es el valor de un coeficiente *DCT bidimensional*, para los índices ***u***, ***v*** de la matriz de coeficientes.

Las ecuaciones anteriores tienen como consecuencia la generación de un conjunto de funciones base bidimensionales, las cuales junto a las muestras en dos dimensiones generan los coeficientes *DCT bidimensionales*. En una imagen las funciones base corresponden a las frecuencias espaciales.

Debido a que las imágenes son arreglos en dos dimensiones, la *DCT* en una dimensión se puede extender para transformar arreglos de dos dimensiones que se generan a partir de la multiplicación de dos conjuntos de una dimensión, un conjunto unidimensional representa las frecuencias espaciales horizontales y el otro las frecuencias espaciales verticales. Por convención el coeficiente de *DC* de las frecuencias espaciales horizontales se encuentra en el extremo izquierdo del arreglo y el coeficiente de las frecuencias base verticales en la parte superior, en consecuencia el único término de *DC*, es el que se encuentra en la esquina superior izquierda.

Para la definición de la *DCT* de 64 muestras, se genera un conjunto de 64 funciones base las cuales se muestran en la ***Figura 3.3***, en está figura se observa la variación de todas las funciones base, así como la función que corresponde al coeficiente de *DC* (esquina superior izquierda) que presenta un valor constante.

Con estas 64 funciones base se puede generar cualquier arreglo de 8x8 coeficientes. La elección de estás dimensiones no solo radica en que resulta extremadamente complejo procesar un gran número de muestras al mismo tiempo, sino que al manejar un bloque pequeño nos permite desarrollar un proceso mas óptimo [3].

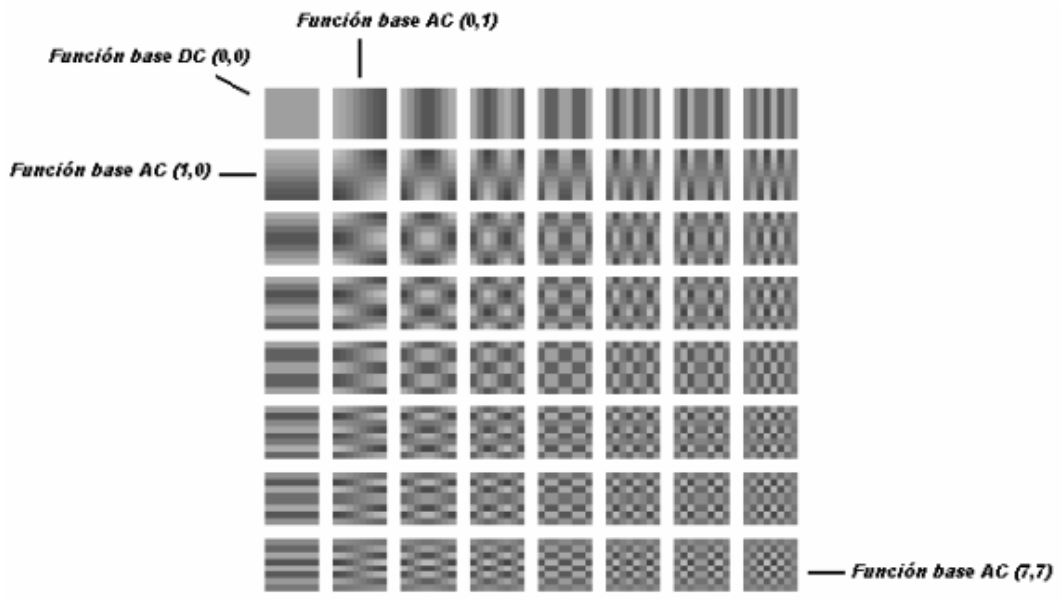


Figura 3.3 .- Funciones base de la *DCT* bidimensional de 8x8

Los valores de la función base (0,0), se pueden encontrar del producto de las dos funciones Coseno, además de ser escalados por los factores ***C(u)***, ***C(v)***; los valores que corresponden a la Función base *DC*, son mostrados en la ***Tabla 3.1***.

0.125	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*
0.125	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*
0.125	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*
0.125	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*
0.125	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*
0.125	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*
0.125	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*
0.125	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*	*0.125*

Tabla 3.1.- Valores Función base DC(0,0)

Para obtener el coeficiente de *DC* es necesario realizar una suma de productos con los elementos de la matriz original que aparecen en la ***Tabla 3.2***, con los valores de la función base *DC*. Esto se realiza multiplicando el elemento (0,0) de la matriz original con el correspondiente elemento (0,0) de la función base, a continuación se realiza la multiplicación de los elementos (0,1) de las matrices original y Función base *DC*, el proceso continua hasta alcanzar los últimos elementos de ambas matrices, y se realiza la sumatoria de todos ellos, el resultado de este proceso es el coeficiente de *DC*.

84	*84*	*90*	*92*	*88*	*66*	*77*	*86*
54	*56*	*72*	*78*	*120*	*124*	*98*	*66*
55	*56*	*80*	*120*	*128*	*98*	*74*	*87*
63	*55*	*120*	*140*	*245*	*124*	*66*	*88*
33	*33*	*45*	*56*	*56*	*87*	*124*	*125*
88	*89*	*90*	*100*	*124*	*43*	*46*	*76*
66	*56*	*82*	*78*	*33*	*128*	*99*	*67*
64	*76*	*74*	*78*	*122*	*121*	*34*	*35*

Tabla 3.2.- Matriz original A(x,y)

Las operaciones correspondientes para obtener el coeficiente *DC*, se desarrollan en las siguientes líneas:

(84x0.125) +(84x0.125) + (90x0.125) + (92x0.125) + (88x0.125) + (66x0.125) + (77x0.125) + (86x0.125)
(54x0.125) +(56x0.125) + (72x0.125) + (78x0.125) +(120x0.125) +(124x0.125) + (98x0.125) + (66x0.125)
(55x0.125) +(56x0.125) + (80x0.125) +(120x0.125) +(128x0.125) + (98x0.125) + (74x0.125) + (87x0.125)
(63x0.125) +(55x0.125) +(120x0.125) +(140x0.125) +(245x0.125) +(124x0.125) + (66x0.125) + (88x0.125)
(33x0.125) +(33x0.125) + (45x0.125) + (56x0.125) + (56x0.125) + (87x0.125) +(124x0.125) +(125x0.125)
(88x0.125) +(89x0.125) + (90x0.125) +(100x0.125) +(124x0.125) + (43x0.125) + (46x0.125) + (76x0.125)
(66x0.125) +(56x0.125) + (82x0.125) + (78x0.125) + (33x0.125) +(128x0.125) + (99x0.125) + (67x0.125)
(64x0.125) +(76x0.125) + (74x0.125) + (78x0.125) +(122x0.125) +(121x0.125) + (34x0.125) + (35x0.125)
= 670.25

Este resultado es el valor del coeficiente *DC*; como puede observarse son necesarias 56 sumas y 64 multiplicaciones para el cálculo de un solo coeficiente.

De la misma forma deben calcularse los coeficientes de *AC*, pero obviamente las operaciones se realizarán con los elementos correspondientes a cada función base, por ejemplo los elementos de la función base (0,1) listados en la ***Tabla 3.3***, nos permiten calcular el primer coeficiente de *AC*.

0.1734	*0.147*	*0.0982*	*0.0345*	*-0.0345*	*-0.0982*	*-0.0147*	*-0.1734*
0.1734	*0.147*	*0.0982*	*0.0345*	*-0.0345*	*-0.0982*	*-0.0147*	*-0.1734*
0.1734	*0.147*	*0.0982*	*0.0345*	*-0.0345*	*-0.0982*	*-0.0147*	*-0.1734*
0.1734	*0.147*	*0.0982*	*0.0345*	*-0.0345*	*-0.0982*	*-0.0147*	*-0.1734*
0.1734	*0.147*	*0.0982*	*0.0345*	*-0.0345*	*-0.0982*	*-0.0147*	*-0.1734*
0.1734	*0.147*	*0.0982*	*0.0345*	*-0.0345*	*-0.0982*	*-0.0147*	*-0.1734*
0.1734	*0.147*	*0.0982*	*0.0345*	*-0.0345*	*-0.0982*	*-0.0147*	*-0.1734*
0.1734	*0.147*	*0.0982*	*0.0345*	*-0.0345*	*-0.0982*	*-0.0147*	*-0.1734*

Tabla 3.3.- Función base AC (0,1)

Para obtener el coeficiente *AC* (0,1) se sigue el mismo procedimiento que en el cálculo del coeficiente de *DC*:

(84x0.1734) +(84x0.147) + (90x0.0982) + (92x0.345) + (88x-0.345) + (66x-0.0982) + (77x-0.147) + (86x-0.1734)
(54x0.1734) +(56x0.147) + (72x0.0982) + (78x0.345) +(120x-0.345) +(124x-0.0982) + (98x-0.147) + (66x-0.1734)
(55x0.1734) +(56x0.147) + (80x0.0982) +(120x0.345) +(128x-0.345) + (98x-0.0982) + (74x-0.147) + (87x-0.1734)
(63x0.1734) +(55x0.147) +(120x0.0982) +(140x0.345) +(245x-0.345) +(124x-0.0982) + (66x-0.147) + (88x-0.1734)
(33x0.1734) +(33x0.147) + (45x0.0982) + (56x0.345) + (56x-0.345) + (87x-0.0982) +(124x-0.147) +(125x-0.1734)
(88x0.1734) +(89x0.147) + (90x0.0982) +(100x0.345) +(124x-0.345) + (43x-0.0982) + (46x-0.147) + (76x-0.1734)
(66x0.1734) +(56x0.147) + (82x0.0982) + (78x0.345) + (33x-0.345) +(128x-0.0982) + (99x-0.147) + (67x-0.1734)
(64x0.1734) +(76x0.147) + (74x0.0982) + (78x0.345) +(122x-0.345) +(121x-0.0982) + (34x-0.147) + (35x-0.1734)
= -57.4938

De esta manera se han calculado solo dos coeficientes *DCT*, siguiendo el mismo procedimiento para todos los coeficientes *DCT* generados por la *Tabla 3.2* se obtiene la matriz de coeficientes *DCT*, la cual se puede observar en la ***Tabla 3.4***.

670.25	*-57.49*	*-106.80*	*26.83*	*28.50*	*-22.82*	*17.18*	*16.74*
35.51	*-20.05*	*-13.05*	*-11.75*	*22.66*	*-0.28*	*-16.88*	*21.21*
-36.07	*53.30*	*25.87*	*19.96*	*-44.43*	*8.89*	*11.50*	*-30.60*
-33.64	*11.90*	*87.80*	*-42.04*	*-9.40*	*26.65*	*-18.71*	*-1.95*
12.5	*-12.85*	*-14.94*	*2.77*	*16.25*	*-23.86*	*0.50*	*17.45*
47.67	*27.36*	*-30.65*	*-12.82*	*15.01*	*6.18*	*13.15*	*3.77*
-0.21	*62.21*	*-11.5*	*-27.55*	*49.52*	*-27.73*	*-19.37*	*8.27*
-56.74	*-55.26*	*82.42*	*-20.57*	*-45.92*	*51.83*	*3.31*	*-44.08*

Tabla 3.4.- Matriz coeficientes *DCT*

El cálculo de la *IDCT bidimensional*, sigue el mismo procedimiento que él de la *DCT*, pero en este caso es necesario utilizar las funciones base que corresponden a la ecuación **(3.15)**.

Gracias a estos dos ejemplos, se puede citar que para el cálculo de la *DCT* e *IDCT bidimensional* para una matriz de *8x8* se requieren un total de *8128* operaciones (*4032 sumas* y *4096 multiplicaciones*).

3.1.6.- Algoritmos optimizados para la FDCT e IDCT

El gran número de operaciones que utilizan la *DCT* e *IDCT bidimensionales,* ha generado que los investigadores desarrollen algoritmos que optimizan el desempeño de la transformada, reduciendo el número de operaciones para su ejecución. Se han creado algoritmos rápidos para el calculo de la *DCT* unidimensional y bidimensional así, como también para la *IDCT* unidimensional y bidimensional.

Como se mencionó en la ***sección 3.1.5***, la *DCT* tiene la propiedad de ser una transformación separable, es decir, se puede obtener una *DCT* bidimensional utilizando la *DCT* unidimensional. Debido a esto, el criterio a utilizar es el implementar un algoritmo rápido para la *DCT* e *IDCT* unidimensionales y con base en ellos obtener sus contrapartes bidimensionales.

Para el caso unidimensional de la *DCT*, el algoritmo optimizado se muestra en la ***Figura 3.4***, en ella entran al proceso ocho muestras y salen ocho coeficientes *DCT*.

Las flechas indican la dirección en que fluye la información, en el caso en que las flechas estén modificadas por un factor la información se multiplicará por el valor de dicho factor, finalmente cuando dos cantidades se encuentran en un nodo estas se deben sumar.

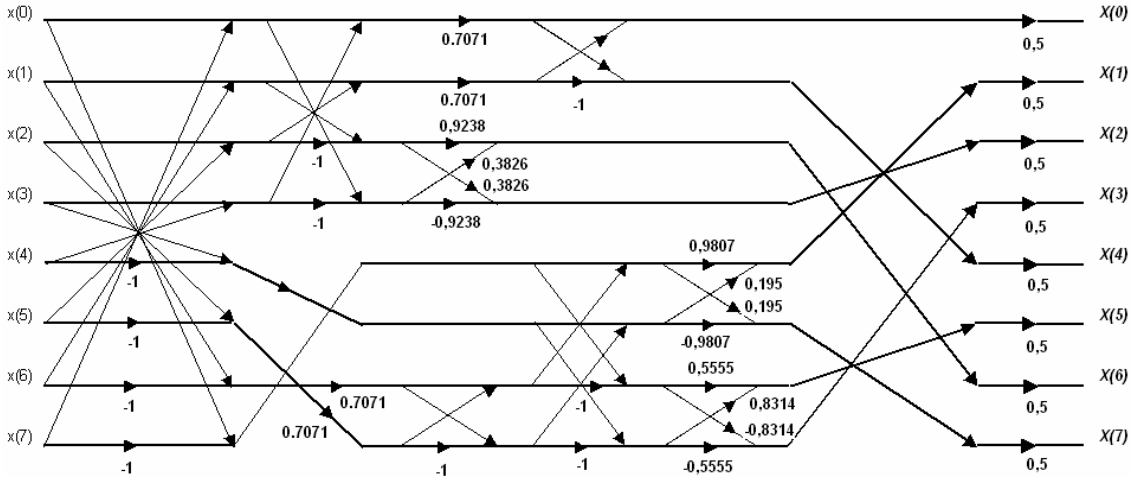


Figura 3.4 .- DCT optimizada algoritmo rápido.

En la ***Figura 3.5*** se muestra el diagrama de flujo que se utiliza para implementar la *IDCT unidimensional* para un conjunto de ocho muestras, estos dos algoritmos rápidos, fueron desarrollados por Wang, Suehiro y Hatori. En la práctica se han creado otras implementaciones como las hechas por Feing, las cuales requieren operaciones de corrimiento [4][1].

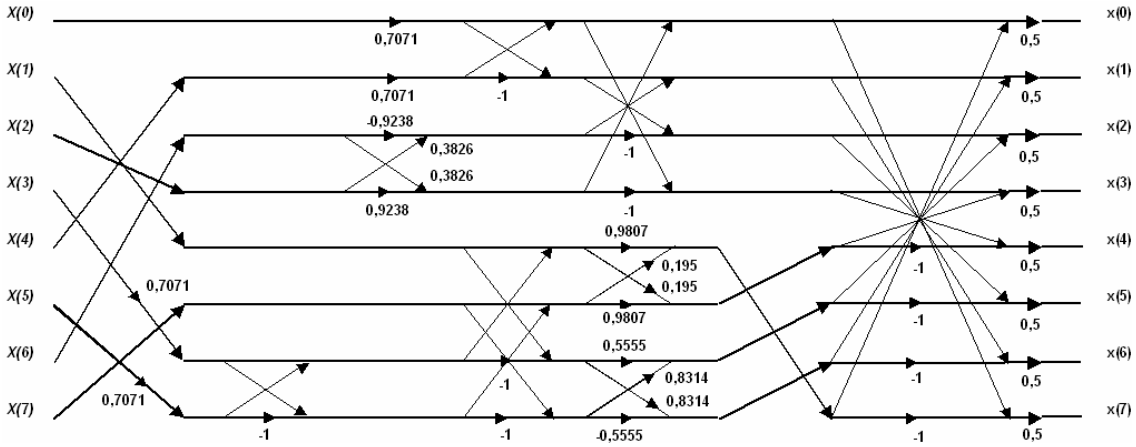


Figura 3.5 .- IDCT optimizada algoritmo rápido.

El procedimiento para obtener una *DCT bidimensional* a partir de la utilización de la *DCT optimizada* es el siguiente, de la matriz original (***Tabla 3.2***) se toman los elementos del primer renglón, como las muestras de entrada al algoritmo rápido con lo que se obtiene a la salida los coeficientes *DCT unidimensionales*, estos coeficientes corresponden al primer renglón de una matriz que para fines convencionales se denominada *intermedia*, después de obtener este primer renglón, se procede a aplicar la *DCT unidimensional* en los renglones siguientes de la matriz original, con lo cual se obtiene la matriz *intermedia procesada por renglones* (***Tabla 3.5***).

235.85	*8.98*	*-3.66*	*-14.39*	*11.66*	*0.01*	*-4.22*	*5.87*
236.20	*-41.88*	*-44.06*	*36.27*	*-11.31*	*-5.26*	*4.47*	*9.47*
246.81	*-28.95*	*-58.14*	*-0.49*	*28.99*	*-5.14*	*1.89*	*-1.67*
318.59	*-28.18*	*-131.61*	*21.81*	*60.46*	*-45.58*	*12.04*	*50.44*
197.66	*-94.61*	*26.02*	*-8.77*	*-6.71*	*14.97*	*-2.74*	*-1.14*
231.96	*34.47*	*-27.33*	*-15.58*	*42.43*	*-23.14*	*-12.40*	*20.53*
215.34	*-26.75*	*-0.35*	*13.83*	*-42.78*	*35.02*	*29.61*	*-29.34*
213.57	*14.33*	*-62.91*	*43.23*	*-2.12*	*-35.41*	*19.94*	*-6.80*

Tabla 3.5.- Matriz Intermedia procesada por renglones

Una vez que se han obtenido todos los renglones de la matriz intermedia, se aplica la *DCT optimizada* tomando como muestras de entrada los elementos de la primera columna de la matriz intermedia, los datos de salida corresponden a la primera columna de la matriz final de coeficientes *DCT*, este proceso continúa hasta que se han procesado todas las columnas de la matriz intermedia. El resultado es la matriz final *DCT* que se da en la ***Tabla 3.6***, la cual debe tener los mismos valores que la obtenida por el método de las funciones base (***Tabla 3.4***).

670.42	*-57.49*	*-106.80*	*26.83*	*28.50*	*-22.82*	*17.18*	*16.74*
35.51	*-20.05*	*-13.04*	*-11.75*	*22.66*	*-0.28*	*-16.88*	*21.47*
-36.07	*53.30*	*25.87*	*19.96*	*-44.42*	*8.89*	*11.49*	*-30.60*
-33.65	*11.90*	*87.79*	*-42.03*	*-9.41*	*26.65*	*-18.71*	*-1.95*
12.50	*-12.85*	*-14.94*	*2.77*	*16.25*	*-23.86*	*0.50*	*17.46*
47.67	*27.36*	*-30.64*	*-12.82*	*15.01*	*6.18*	*13.15*	*3.76*
-0.20	*62.20*	*-11.49*	*-27.55*	*49.52*	*-27.72*	*-19.37*	*8.27*
-56.75	*-55.26*	*82.41*	*-20.58*	*-45.93*	*51.83*	*3.31*	*-44.08*

Tabla 3.6.- Matriz Final DCT *F(u,v)*

Las matrices de las ***Tablas 3.4*** y ***3.6*** son idénticas en teoría (aunque realmente difieren debido a que la precisión utilizada por el algoritmo rápido es menor). En este ejemplo la *DCT optimizada* se aplicó primero a los renglones y luego a las columnas, el orden de aplicación es arbitrario pues también es posible procesar primero las columnas y después los renglones. Para obtener una *IDCT bidimensional* se sigue el mismo procedimiento utilizando el algoritmo ***Figura 3.5***.

Utilizando los algoritmos optimizados tanto para la *DCT*, como para la *IDCT* es posible reducir de manera considerable el número de operaciones, por ejemplo :

- Para la *DCT unidimensional* optimizada (algoritmo de Wang, Suehiro, Hatori) solo se necesitan ***26 sumas y 16 multiplicaciones***, para un conjunto de ocho muestras, reduciendo el porcentaje de operaciones hasta el *35%.*

- Para la *IDCT unidimensional optimizada* se necesitan un total de ***27 sumas y 16 multiplicaciones***, para un conjunto de ocho muestras, reduciendo el porcentaje de operaciones hasta el *35.83%.*

- Utilizando el criterio descrito en está sección para el cálculo de la *DCT bidimensional*, solo se necesitan un total de ***416 sumas y 256 multiplicaciones***, para un bloque de 8x8, reduciendo el porcentaje de operaciones hasta el *8.267%.*

- De igual manera para obtener la *IDCT bidimensional* se necesitan ***432 sumas y 256 multiplicaciones***, para un bloque de 8x8, reduciendo el porcentaje de operaciones hasta el *8.464%*.

Esta descripción de la *DCT* e *IDCT* es necesaria debido a que los algoritmos de compresión y descompresión *MJPEG*, hacen uso amplio de estas transformaciones.

Puesto que la secuencia de video que se desea codificar, está formada por arreglos bidimensionales (***Sección 1.3***), es preciso encontrar un procedimiento que permita implementar las transformaciones con la menor cantidad de operaciones posibles, a su vez la gran cantidad de sumas y multiplicaciones que tienen lugar por el uso de estas transformadas fue la razón principal que motivó a la utilización de un *DSP* como herramienta principal para este trabajo.

3.2.- Estándar MJPEG (Motion JPEG)

Como se mencionó en la ***sección 2.3.5***, el estándar *MJPEG* está basado en la codificación intra *JPEG* para cada imagen independiente contenida en una secuencia de video, debido a esto, todas las características aplicables a *JPEG* lo serán para *MJPEG*. *JPEG* es un estándar internacional para compresión de imágenes en escala de grises y de color de alta calidad (Tonos continuos), la motivación principal para el desarrollo de *JPEG* fue promover el uso del color en las imágenes con calidad fotográfica, considerando el hecho que una imagen en color digitalizada puede tener 24 bits por píxel, 8 bits para cada componente de *RGB* o *YCrCb* [10].

El grupo *JPEG* realizó el desarrollo de algoritmos para imágenes con razones mayores de un bit por píxel, para realizar esto se utilizan métodos de compresión con pérdidas y sin pérdidas.

Los objetivos fundamentales de *MJPEG* son [1][10]:

- Ser aplicable prácticamente a cualquier tipo de imágenes fuente digitales en tonos continuos, no debe ser restringido a imágenes de ciertas dimensiones, espacios de color, razón de aspecto y no debe ser limitado a cierta clase de imágenes con restricciones de contenido de escena, tales como complejidad, intervalo de colores o propiedades estadísticas.

- Debe tener complejidad computacional manejable para realizar implementaciones de software adecuadas sobre un amplio intervalo de arquitecturas, también sus implementaciones en hardware serán adecuadas para aplicaciones que requieren de alto desempeño.

Para reducir el número de bits usados para representar una secuencia de video digitalizada *MJPEG* toma ventaja de la redundancia e irrelevancia, (***secciones 2.2*** a ***2.2.3***), al aplicarse el algoritmo *MJPEG* en cada imagen independiente de la secuencia de video, se estará eliminando dos de los tres tipos de redundancia presente en el video digital (redundancia espacial e intersímbolos).

La redundancia espacial se incrementa con resoluciones altas de la imagen, debido a que las regiones de la imagen están representadas por mas píxeles, para la realización de este tipo de eliminación *JPEG* se basa en estudios realizados del sistema visual Humano, la codificación Perceptiva (***Sección 2.2***) toma ventajas de varias limitaciones del ojo humano incluyendo imperfecciones en la respuesta del color, el ojo humano es menos sensible a algunos componentes que otros.

Es menos sensible a la crominancia que a la luminancia, esto implica que pocos bits son necesarios para la información de crominancia, pudiéndose codificar más bruscamente que la Luminancia, otro fundamento de la codificación perceptiva es tomar ventaja de que el ojo es mas sensible a las frecuencias espaciales bajas [1][14][16][18]. Esto permite efectuar diseños basados en la fidelidad de los contornos donde ocurren transiciones rápidas en el brillo, también el ojo es menos sensible a las distorsiones en los niveles altos de luminancia.

Así mismo el estándar *MJPEG* describe una familia de técnicas de compresión de imágenes, lo que provee un “estuche de herramientas” de técnicas de compresión para aplicaciones que pueden seleccionar elementos que satisfacen diferentes requerimientos particulares.

Por ejemplo las imágenes que *MJPEG* soporta pueden estar en el intervalo de *1x1* o *65536x65536* píxeles, cada píxel puede tener de *1* a *255* componentes en color o bandas espectrales.

Debido ha estos requerimientos se provee de los siguientes modos de operación [1][14]:

- Codificación secuencial, basada en la *DCT.*
  - Modallidad básica
- Codificación progresiva, basada en la *DCT.*
- Codificación sin perdidas.
- Codificación jerárquica.

Estos cuatro modos de operación, han resultado de la meta de *JPEG* de ser genérico y de la diversidad de formatos de imagen en las aplicaciones. Las múltiples piezas pueden dar la impresión de una complejidad indeseable, pero como ya se mencionó se debe considerar como “un estuche de herramientas”, las cuales pueden soportar un amplio intervalo de aplicaciones de imágenes en tono continuo y en movimiento.

3.2.1.- Modos de operación

Los modos basados en la *DCT* proveen compresión con pérdidas, mientras que una forma predictiva de Codificación por Modulación Diferencial de Pulso (*DPCM*) es usada por el modo secuencial sin pérdidas, el modo jerárquico puede usar extensiones basadas en la *DCT* o en la codificación predictiva [1][14].

La ***Figura 3.6*** muestra un esquema de esos cuatro modos y algunas de las relaciones que existen entre ellos.

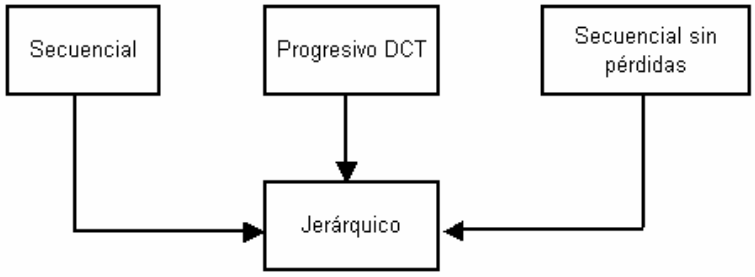


Figura 3.6 .- Modos de operación JPEG.

Existen muchas similitudes y elementos comunes en estos modos, estas similitudes de conceptos y estructuras de codificación hacen al sistema total mucho menos complejo que el que parece a primera vista.

Las imágenes en tono continuo pueden ser comprimidas por una razón de *2:1* con las técnicas *MJPEG* sin pérdidas, sin embargo las técnicas de compresión con pérdidas pueden alcanzar razones de *20:1.*

3.2.1.1.- Modo de operación secuencial basado en la DCT

El *modo de operación secuencial basado en la DCT* codifica los componentes de la imagen, en un simple "barrido" de *izquierda a derecha* y de *arriba hacia debajo* de la imagen. La manera en que esto se realiza, es agrupando la imagen en bloques de *8x8* (empezando en el extremo superior izquierdo llegando hasta el extremo inferior derecho), esta manera de procesar la imagen permite que cada bloque de *8x8* píxeles, pueda ser cuantificado y codificado inmediatamente después de haber sido transformado minimizando los recursos de memoria utilizados.

Debido a que los datos fueron codificados en un simple barrido, el decodificador que reconstruya la imagen debe seguir este mismo orden, esto se puede apreciar en la ***Figura 3.7***, que muestra el orden de decodificación de una imagen empezando en la esquina superior izquierda y terminado en la inferior derecha [1][16].

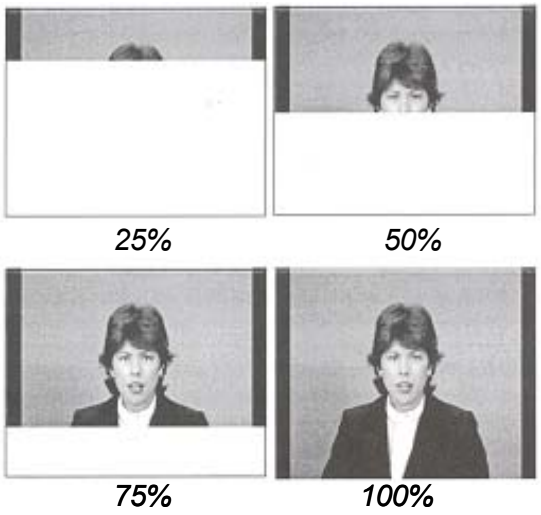


Figura 3.7 .- Modo de operación Secuencial

Una forma particular de restricción del *modo de operación secuencial basado en la DCT* es conocida como *"modalidad básica o línea base",* el cual debe estar presente en todos los sistemas de decodificación basados en la *DCT*. La *modalidad básica* utiliza solo 8 bits por componente de espacio de color, de 1 a 4 componentes por espacio de color (*YCrCb, RGB, etc*), los datos de los componentes pueden presentarse intercalados o separados (***sección 3.3.4***) y solo se podrá aplicar la codificación Huffman (***secciones 3.3.4.4*** y ***3.3.4.6***). La *modalidad básica* es el fundamento en el cual se centra el desarrollo de este trabajo.

3.2.1.2.- Modo de operación progresivo basado en DCT

En la *modalidad progresiva* los datos se procesan en el mismo orden que en la *modalidad secuencial*, sin embargo en este caso ocurren varías lecturas de información de una imagen, los coeficientes *DCT*, se codifican parcialmente en cada lectura, existen dos procedimientos para codificar parcialmente los coeficientes [1][16].

- Al primero se le llama selección espectral debido a que los coeficientes de un bloque se dividen dependiendo de su frecuencia espacial baja, siguiéndolos los coeficientes con más altas frecuencias.

- Al segundo procedimiento se le conoce como aproximaciones sucesivas, en la primera lectura de datos se codifican los bits más significativos de todos los coeficientes y en las lecturas posteriores se codifican los bits menos significativos.

Por ejemplo los coeficientes *DC* son codificados primero, generando que la imagen se reconstruya en bloques, el primer barrido de coeficientes *AC* incluye solo los dos coeficientes de menor frecuencia, esto remueve mucho del efecto de bloques que aparece en el primer barrido, la calidad de la imagen es significativamente mejorada después de que cinco coeficientes de *AC* son codificados, cuando todos los coeficientes son codificados el proceso termina.

En la ***Figura 3.8*** se observa el procedimiento de reconstrucción por el *modo progresivo*. Primero se decodifica una aproximación de la imagen entera (decodificación del coeficiente *DC*) y luego se decodifican los detalles finos para obtener la imagen completa con cada barrido sucesivo (decodificación de los coeficientes de *AC*); esta forma particular de compresión progresiva definida por *JPEG* puede proveer una imagen final que es idéntica a la del modo secuencial. Normalmente este modo utiliza de cuatro a seis barridos para los procesos de codificación y decodificación.

La aproximación sucesiva da mejores calidades con razones de bits mas bajas, en la aproximación sucesiva los coeficientes de *DCT* son representados con diferentes precisiones en cada barrido, estos dos procesos para el método progresivo pueden ser entremezclados para proveer un desempeño más óptimo.

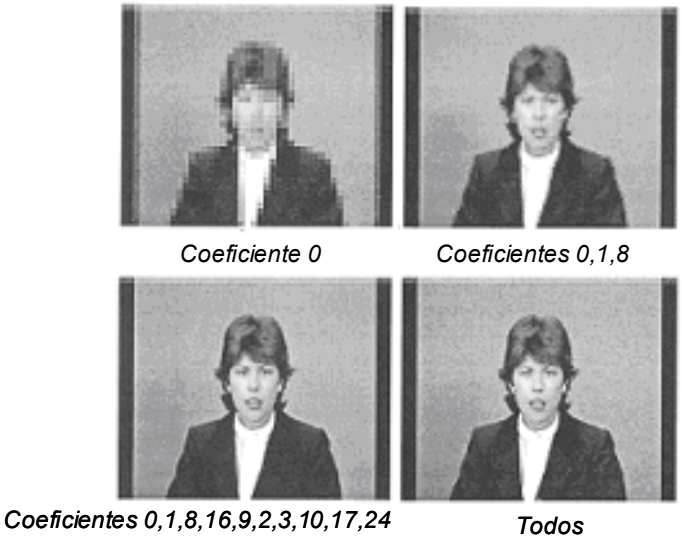


Figura 3.8 .- Modo de operación Progresivo

3.2.1.3.- Modo de operación Jerárquico

Una forma alternativa de codificación progresiva es conocida como *Modo jerárquico*, el cual utiliza un conjunto de imágenes sucesivas más pequeñas que son creadas por submuestreo, entonces el proceso de codificación inicia con la imagen más pequeña, usando uno de los *modos secuencial o progresivo JPEG*. Después de eso se procede a codificar las imágenes sucesivas, cuando el conjunto de imágenes es almacenado asemeja a una pirámide, consecuentemente esta forma de codificación es conocida también como codificación piramidal.

En la ***Figura 3.9***, se observa la decodificación utilizando el *modo jerárquico*, en esta figura se muestra como la imagen original es generada a partir de un conjunto de sub-imágenes, dichas sub-imágenes son decodificadas y nos permiten reconstruir la original.

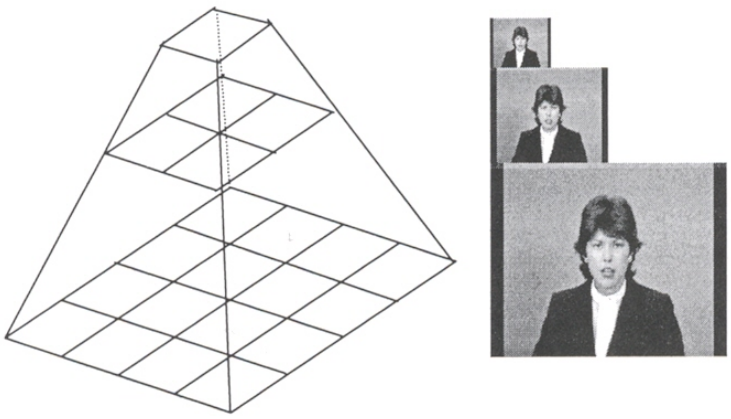


Figura 3.9 .- Modo de operación Jerárquico

3.2.1.4.- Modo Secuencial sin perdidas

Una representación exacta entre la entrada del codificador y la salida del decodificador no puede ser garantizada entre las diferentes modos del sistema *JPEG* basados en *DCT*, debido a que los datos reconstruidos por la *IDCT* no son iguales a los originales [1].

El modo de operación sin pérdidas usa una técnica de codificación predictiva que es una extensión del modelo de codificación usado para los coeficientes *DC* (***Sección 3.3.4.4***) en los modos que utilizan la *DCT*.

Suponiendo que se tiene una imagen que necesita ser codificada y no se desea pérdida de información, se podría alimentar simplemente los valores de las muestras al codificador entrópico (eliminación de redundancia estadística) este es el método de codificación más simple posible y también es uno de los métodos más pobres, conocido como Modulación de Codificación por Pulso (*PCM*) [1].

Un modelo mucho mejor, es realizado si se intenta predecir el valor de las muestras de una combinación de los elementos ya codificadas en la imagen (el decodificador también debe conocer aquellos valores, para que pueda hacer la misma predicción), por ejemplo asumiendo que se está codificando de izquierda a derecha, podríamos usar la muestra de la izquierda como una estimación para las muestras siguientes, entonces el valor que se alimenta al codificador entrópico, puede ser la diferencia entre las muestras y la muestra de la izquierda.

Por lo tanto en una imagen en tonos continuos, las diferencias entre una muestra y la siguiente son comúnmente mas pequeñas y es más eficiente codificar las diferencias, que codificar cada muestra independientemente, esta clase de modelos de codificación es conocido como (*DPCM*).

En la ***Figura 3.10*** se muestra la gráfica de una línea perteneciente a una imagen digital en tonos de grises, en ella se puede observar como sus muestras presentan valores mucho mayores a cero.

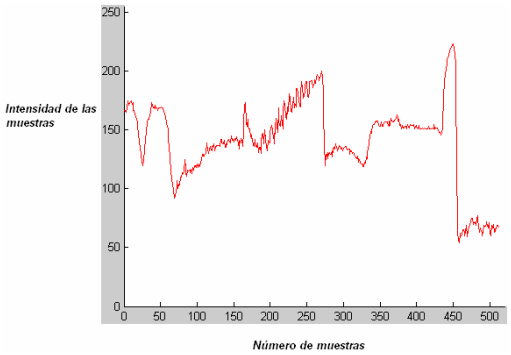


Figura 3.10 .- Valores de Intensidad de una línea típica de imagen digital

La ***Figura 3.11*** muestra la diferencia correspondiente entre cada muestra y la muestra inmediatamente a su izquierda, para la línea mostrada en la ***Figura 3.10***, estos valores diferencia están usualmente mucho mas cercanos a cero.

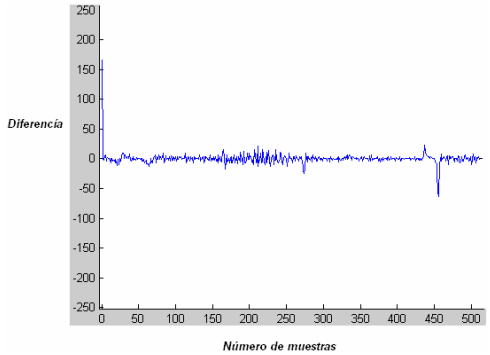


Figura 3.11 .- Valores diferencias

Dado un valor conocido de inicio, las diferencias de una muestra a la siguiente pueden ser usadas para reconstruir exactamente el valor de la intensidad original, por lo tanto el conjunto de diferencias es una representación que es exactamente equivalente a los valores de intensidad original; sin embargo la forma *DPCM*, necesita de menos bits para su representación pues la mayoría

de sus valores están cercanos a cero, el mejor desempeño de *DPCM* relativo a *PCM* es debido al alto grado de correlación encontrado en la mayoría de las imágenes. Esta idea es el criterio principal utilizado en el modo secuencial sin pérdidas, y también es utilizado para la codificación de los coeficientes *DC*, en el modo básico (***Sección 3.3.4.3***).

En la práctica solo los *modos secuencial, básico* y *progresivo* son utilizados, estos modos utilizan la codificación entrópica Huffman, necesitando de tablas de códigos que deben ser conocidas al inicio de los procesos de codificación y decodificación.

3.2.1.5.- Formato de datos comprimidos de Intercambio y abreviado

Existen tres formatos para los datos *MJPEG* comprimidos [1]:

- El *Formato de intercambio* incluye todos los datos codificados entrópicamente, así como todas las tablas que son requeridas por el decodificador, si los datos comprimidos de video son manejados para uso general, el formato de intercambio debe ser usado.

- El *Formato abreviado para datos comprimidos* puede omitir algunas o todas las tablas necesarias para la decodificación, esas tablas deben ser añadidas previamente al flujo de datos comprimidos o se deben conocer a través de algún otro mecanismo, por el decodificador.

- *El formato abreviado para una tabla de especificación de datos* no contiene segmentos codificados entrópicamente, este formato abreviado es usado para establecer las tablas que pueden ser añadidas al *formato abreviado para datos comprimidos*.

3.2.1.6.- Estructura de datos comprimidos

Los datos comprimidos por *MJPEG* contienen dos tipos de datos [1]:

- Los segmentos *codificados entrópicamente.*

- Los segmentos “marcadores”.

Los segmentos *codificados entrópicamente* son los datos codificados, mientras que los segmentos marcadores contienen la información de cabecera como tablas y otra información requerida para interpretar y decodificar los datos comprimidos de una secuencia de video. Los segmentos marcadores siempre empiezan con un código único de un byte que identifica la función del segmento.

3.2.1.7.- Definición de los marcadores

Cada marcador de segmento comienza con el código hexadecimal *FF* seguido del código de marcador que identifica la función que realiza, los marcadores caen en dos categorías: aquellos sin parámetros y aquellos seguidos por una secuencia de longitud variable de parámetros de estructura conocida. Entre los marcadores mas importantes definidos por el estándar *MJPEG* se encuentran los siguientes [1]:

DHP: Define Progresión Jerárquica, es usado para identificar las dimensiones de los parámetros de señal, componentes y muestreo relativo de la imagen final en la codificación jerárquica.

DHT: Define una o más tablas de Huffman utilizadas.

DQT: Define una o mas tablas de cuantización, este marcador solo se define para los algoritmos basados en la *DCT*.

EOI: Fin de la Imagen, indica el fin del flujo de datos comprimidos *MJPEG*, para un cuadro de la secuencia de video.

SOI: Inicio de Imagen, indica el comienzo del flujo de datos comprimidos para un cuadro de la secuencia de video.

3.3.- Modalidad Básica del estándar MJPEG

Como se señaló en la ***sección 3.2.1.1***, el modo secuencial basado en *DCT* estipula una modalidad básica la cual es la base de este trabajo, el objetivo de esta sección es describir está modalidad, para comprender como se implementa la codificación y decodificación de la secuencia de video, por medio del *DSP* [1].

3.3.1.- Espacios de color utilizados por la modalidad básica

El estándar *MJPEG* no especifica que modelo de color se debe utilizar, sin embargo para imágenes en color los algoritmos de compresión tienen un mejor desempeño al utilizar un sistema de coordenadas basados en luminancia-crominancia, la componente que cuenta con más cambios en su frecuencia espacial es la de luminancia, los componentes de crominancia casi no presentan cambios de tono, consecuentemente la compresión para los componentes de crominancia es grande.

En contraste con un espacio *RGB* los cambios de tono se reflejan en todas las componentes, por lo tanto no se podrá compactar la información tanto como en el caso de un modelo basado en luminancia-crominancia, debido a lo anterior el modelo *YCrCb* es el más utilizado en las aplicaciones *MJPEG* para las imágenes en color, incluyendo la modalidad básica [6].

3.3.2.- Unidad mínima de codificación (MCU)

La *unidad de datos* es la unidad lógica más pequeña proveniente de la fuente de datos original, que puede ser procesada en un modo de operación *MJPEG*, en el modo sin pérdidas los arreglos son procesados una muestra a la vez, por lo tanto la *unidad de datos* para procesamiento sin pérdidas es una muestra, por definición las muestras son procesados de izquierda a derecha para los columnas y del renglón superior al inferior para cada componente.

En los modos basados en la *DCT*, los arreglos para componentes son divididos en bloques de *8x8* para propósitos de cómputo de la *DCT* (***sección 3.1.5***), por lo tanto la *unidad de datos* para un proceso basado en la *DCT* es un simple bloque de *8x8* muestras. Por definición los bloques, son procesados de izquierda a derecha y de arriba hacia abajo.

Para los procedimientos de codificación las *unidades de datos* son ensambladas en grupos llamados *"unidades mínimas de codificación (MCU´s)"*, en los barridos con solo un componente la *MCU* es solo una unidad de datos. En barridos con más de una componente sus *unidades de datos* serán agrupadas en una sola *MCU*. El número de *unidades de datos*, en una *MCU* dependerá de las resoluciones espacial y vertical de cada componente.

La codificación entrópica siempre es realizada en una *MCU* completa, por lo tanto en el codificador cualquier *MCU* incompleta debe ser completada por la réplica de la columna más a la izquierda y/o el último renglón de cada componente, cualquier columna extra y renglón añadido por el codificador es descartado por el decodificador [1][16].

3.3.3.- Codificación de datos intercalados y no intercalados.

La información contenida en una imagen de múltiples componentes puede ser agrupada en una *MCU* de dos maneras distintas [1]:

- *Codificación entrelazada*.
- *Codificación no entrelazada*.

Tomando en cuenta la ***Figura 3.12***, la imagen cuenta con tres componentes A, B, C si se procesan primero todas las *unidades de datos* de la componente A, después las de B y al último las de C, su codificación será *no entrelazada*, para el caso particular en que todas las componentes tengan las misma resoluciones se codificará una *unidad de datos* de la componente A, una de B y una de C, para volver a la siguiente unidad de A y así sucesivamente obteniendo una *codificación entrelazada*.

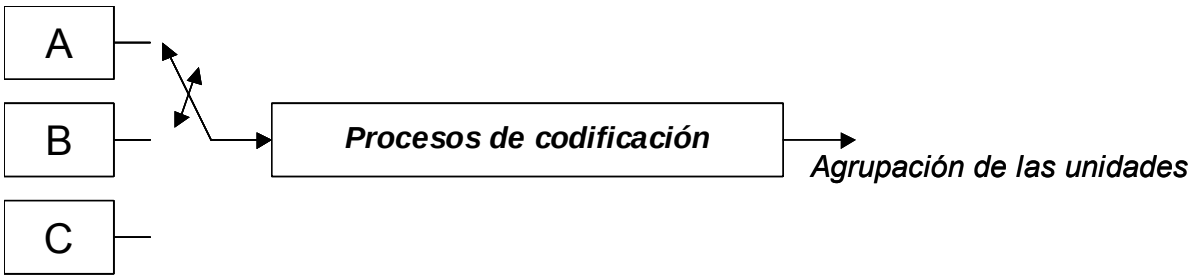


Figura 3.12.- Codificación de las unidades de datos

Las tablas de cuantización y codificación (***Secciones 3.3.4.2, 3.3.4.4*** y ***3.3.4.6***) pueden ser diferentes para las componentes, además es posible intercalar componentes con distinta resolución. La ***Figura 3.13*** muestra 24 unidades de datos (ocho para cada componente), provenientes de la imagen que se desea codificar; el orden de agrupamiento *no entrelazado* se muestra en la ***Tabla 3.7***, mientras que el entrelazado se aprecia en la ***Tabla 3.8***.

Figura 3.13.- Unidades de datos

Como se puede apreciar en la ***Tabla 3.7***, se realizan tres barridos lo que da lugar a la generación de *24 MCU´s* (ocho para cada barrido), mientras que en la ***Tabla 3.8***, solo se realiza un barrido, con lo que se obtienen *8 MCU´s* en total.

Unidad de datos	*MCU*
Barrido 1	
Y1	*1*
Y2	*2*
Y3	*3*
Y4	*4*
Y5	*5*
Y6	*6*
Y7	*7*
Y8	*8*
Barrido 2	
Cr1	*9*
Cr2	*10*

Tabla 3.7.- Datos no entrelazados

Unidad de datos	*MCU*
Barrido 1	
Y1	*1*
Cb1	*1*
Cr1	*1*
Y2	*2*
Cb2	*2*
Cr2	*2*
Y3	*3*
Cb3	*3*
Cr3	*3*

Tabla 3.8.- Datos entrelazados

Cr3	*11*	*Y4*	*4*
Cr4	*12*	*Cb4*	*4*
Cr5	*13*	*Cr4*	*4*
Cr6	*14*	*Y5*	*5*
Cr7	*15*	*Cb5*	*5*
Cr8	*16*	*Cr5*	*5*
Barrido 3		*Y6*	*6*
Cb1	*17*	*Cb6*	*6*
Cb2	*18*	*Cr6*	*6*
Cb3	*19*	*Y7*	*7*
Cb4	*20*	*Cb7*	*7*
Cb5	*21*	*Cr7*	*7*
Cb6	*22*	*Y8*	*8*
Cb7	*23*	*Cb8*	*8*
Cb8	*24*	*Cr8*	*8*

Tabla 3.7.- Datos no entrelazados (continuación)

Tabla 3.8.- Datos entrelazados (continuación)

3.3.4.- Descripción de la Modalidad Básica del codificador MJPEG

La ***Figura 3.14*** muestra el esquema a bloques del codificador de vídeo *MJPEG* para el modo básico, en cada cuadro de la secuencia de video original es aplicado el algoritmo *JPEG* (así como para cada componente de un cuadro), generando como salida los datos comprimidos de la secuencia de video.

Este algoritmo tiene como partes principales las etapas de corrimiento, la *DCT* (***Sección 3.1***), cuantización y codificación entrópica que se describirán en los siguientes apartados.

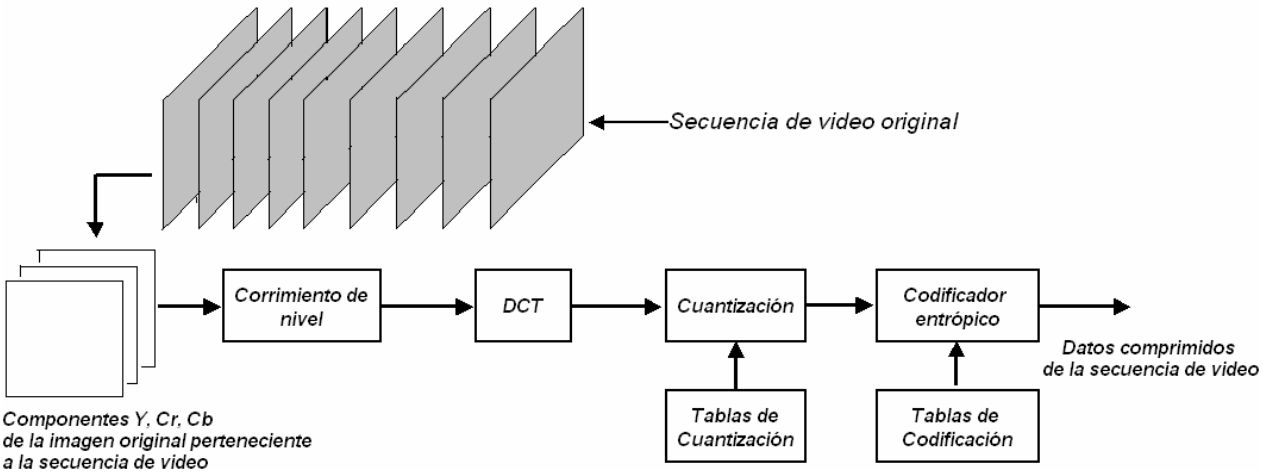


Figura 3.14.- Codificador Modo Básico para MJPEG

En la ***Figura 3.15*** se muestra el esquema correspondiente al decodificador *MJPEG* para una secuencia de video, en este caso se tiene la entrada de datos comprimidos, generando como salida la secuencia de video reconstruida.

Como se puede ver, el decodificador posee el mismo número de etapas que el codificador, siendo una de sus mayores ventajas, ya que provee simetría al sistema de compresión-decompresión con respecto a otros [1][16].

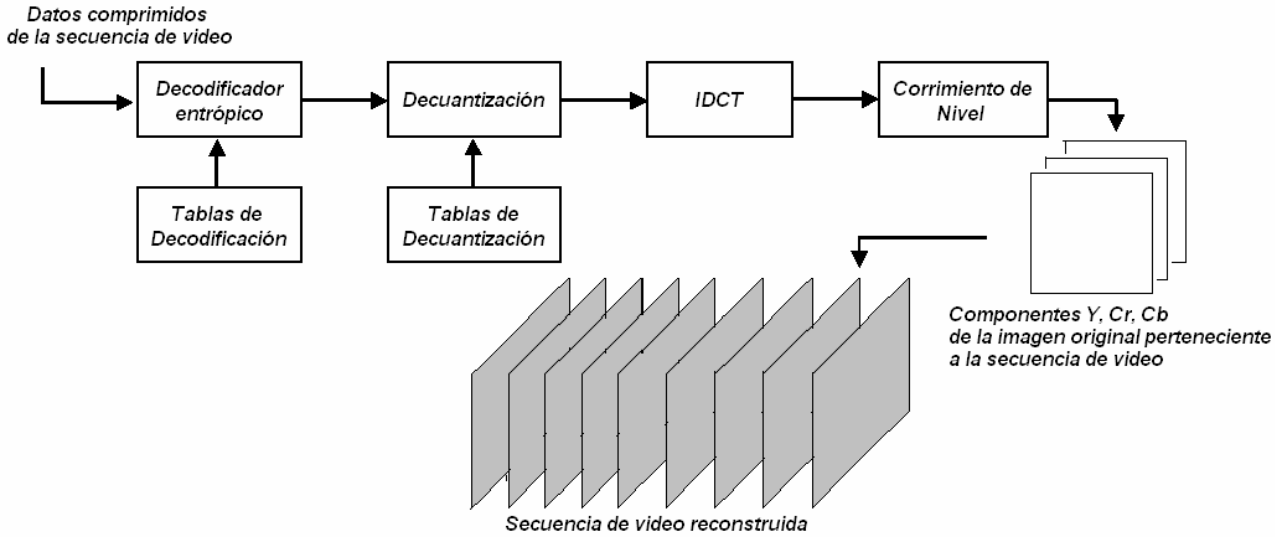


Figura 3.15.- Decodificador modo básico para MJPEG

3.3.4.1.-Corrimiento de nivel

Una vez que la imagen original es convertida al espacio de color *YCrCb*, el estándar *MJPEG* estipula que debe realizarse un corrimiento de nivel de los datos que componen una imagen antes de aplicar la *DCT*, dicho corrimiento depende del nivel de precisión numérica *P* con que cuentan las muestras; debido a que el modo básico solo permite componentes de ocho bits (representación entera no signada de 0 a 255), el corrimiento está dado por $2^{(P-1)}$, cuyo resultado es 128, por lo tanto a cada muestra se les restará la constante 128 (obteniendo una representación signada -128 a 127).

En principio este corrimiento afecta solo a los coeficientes *DC* generados por la *DCT*. Después del desarrollo de la *IDCT* los datos de salida son desplazados de nivel por la suma de $2^{(P-1)}$, convirtiendo la salida a una representación no-signada nuevamente [1].

3.3.4.2.- Cuantización

Una vez que son obtenidos los coeficientes *DCT* de la *unidad de datos* desplazada, es posible realizar la eliminación de redundancia espacial, esto se logra en la etapa denominada cuantización. Este proceso de eliminación se lleva a cabo por la división de los elementos de la matriz de coeficientes *DCT*, entre los elementos de las matrices conocidas como tablas de cuantización, su fórmula matemática aparece en la ecuación **(3.16)** [1][16].

$$Matriz\_cuantizada\ (u,v) = Redondeo\left(\frac{\frac{Matriz\_transformada\ (u,v)}{Tabla\_cuantización\ (u,v)}}{Factor\ DIV}\right) \qquad \textbf{(3.16)}$$

Si se desea obtener un grado de compresión diferente es necesario utilizar otros valores para estas tablas, esto se logra escalando los valores de las Tablas de cuantización por el *factor DIV*, el estándar *MJPEG* recomienda que los valores de este factor estén entre *0.2* y *3* [1].

La cuantización está diseñada para reducir los componentes de frecuencia no esenciales, redondeando los valores pequeños a cero. Frecuentemente los valores de las tablas de cuantización (***Tablas 3.9*** y ***3.10***) para la luminancia y crominania son muy diferentes, esto se debe a las limitaciones en la respuesta visual del ojo humano, estas tablas son construidas tomando como base distintos tipos de parámetros, pero en general se considera que las bajas frecuencias son mas importantes que las altas y que la luminancia es mas importante que la crominancia, su obtención queda fuera de este estudio.

16	*11*	*10*	*16*	*24*	*40*	*51*	*61*
12	*12*	*14*	*19*	*26*	*58*	*60*	*55*
14	*13*	*16*	*24*	*40*	*57*	*69*	*56*
14	*17*	*22*	*29*	*51*	*87*	*80*	*62*
18	*22*	*37*	*56*	*68*	*109*	*103*	*77*
24	*35*	*55*	*64*	*81*	*104*	*113*	*92*
49	*64*	*78*	*87*	*103*	*121*	*120*	*101*
72	*92*	*95*	*98*	*112*	*100*	*103*	*99*

Tabla 3.9.- Cuantización de luminancia

17	*18*	*24*	*47*	*99*	*99*	*99*	*99*
18	*21*	*26*	*66*	*99*	*99*	*99*	*99*
24	*26*	*56*	*99*	*99*	*99*	*99*	*99*
47	*66*	*99*	*99*	*99*	*99*	*99*	*99*
99	*99*	*99*	*99*	*99*	*99*	*99*	*99*
99	*99*	*99*	*99*	*99*	*99*	*99*	*99*
99	*99*	*99*	*99*	*99*	*99*	*99*	*99*
99	*99*	*99*	*99*	*99*	*99*	*99*	*99*

Tabla 3.10.- Cuantización de crominancia

El paso de la cuantización es una operación con pérdidas, después de la cual solo una aproximación de los píxeles de los bloques de *8x8* pueden ser obtenida, esto es la causa de la distorsión en las imágenes reconstruidas. La decuantización se realiza efectuando la multiplicación de los elementos de las tablas de cuantización por los de las *unidades de datos* decodificadas.

3.3.4.3.- Codificación de los coeficientes de DC

Una vez transformada y cuantizada la información, ésta se prepara para codificarse entrópicamente, separando los coeficientes de *DC* de los de *AC*, esto se debe a que un coeficiente de *DC* representa el valor promedio de los 64 píxeles en una *unidad de datos* y su nivel de correlación con el coeficiente de *DC* del siguiente bloque de *8x8* será grande, mientras que la mayoría de los coeficientes de *AC* serán cero.

A los coeficientes de *DC* se les aplica el método *DPCM*, en lugar de codificar cada coeficiente por separado, esto da como resultado que se codifiquen las diferencias entre dos coeficientes de *DC*. La diferencia entre dos coeficientes de *DC* se obtiene sustrayendo al coeficiente i-ésimo, el coeficiente que le precede (que pertenece a otro bloque de *8x8*) del mismo componente; al comenzar una lectura de datos de una imagen, el valor que le precede al primer coeficiente *DC* se inicializa con valor cero, por lo que la primera diferencia será el valor del primer coeficiente de *DC*, este procedimiento se ilustra en la ***Figura 3.16***.

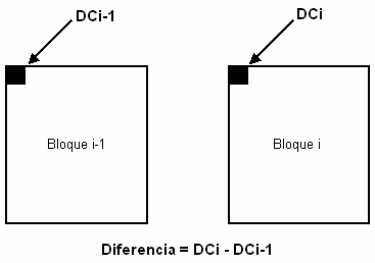


Figura 3.16.- Generación de las diferencias

La ***Figura 3.17*** muestra el diagrama para la obtención de las diferencias *DPCM*, el esquema es muy simple y en él se observa como sus dos únicas entradas son el Coeficiente *DC* y su valor anterior.

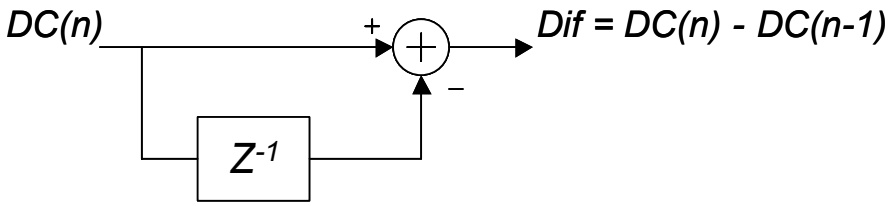


Figura 3.17.- Esquema del codificador *DPCM*.

El decodificador *DPCM* es exactamente la inversa del codificador *DPCM*. En la ***Figura 3.18*** la diferencia *DPCM* es sumada junto con el coeficiente *DC* del bloque *8x8* previo para el mismo componente, el resultado es el coeficiente *DC* del bloque que se está decodificando [1][16].

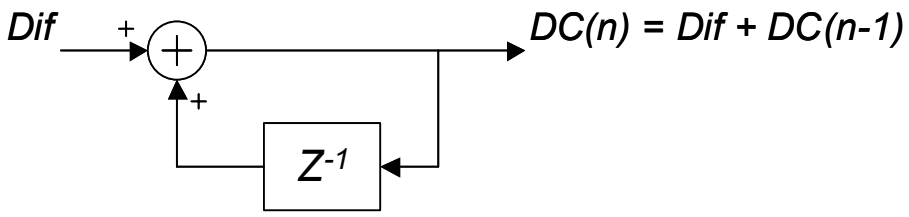


Figura 3.18.- Esquema del decodificador *DPCM*.

3.3.4.4.- Modelo estadístico para codificación de las diferencias DPCM

Dos modelos estadísticos son usados en el modo básico, uno aplica a la codificación de las diferencias *DC* generadas por el modelo *DPCM* y el otro aplica a la codificación de los coeficientes *AC*. Inmediatamente después de usar *DPCM* en los coeficientes *DC*, éstos se ordenan en un modelo estadístico de Huffman que les permitirá ser codificados entrópicamente, dicho modelo estadístico divide las diferencias *DPCM* en categorías, cuyas magnitudes aumentan en forma cuasi-logaritmica [1][16].

Cada una de las categorías diferencia representa un símbolo y por lo tanto le es asignado un código de Huffman (a veces denominado código base), el modelo tiene un alfabeto muy pequeño de símbolos. Excepto para diferencias cero, los códigos base (*categoría SSSS*) no describen totalmente las diferencias *DPCM*, por lo tanto, inmediatamente seguido a estos códigos base para categorías

diferencia de valor diferente de cero, *SSSS bits adicionales* son concatenados a la cadena del código Huffman para identificar el signo y especificar ampliamente la magnitud de la diferencia.

Existe una excepción a esto cuando la *categoría SSSS = 16* no es seguido por ningún bit adicional (al igual que *SSSS = 0*). La regla para concatenar *SSSS bits adicionales* a sus correspondientes códigos Huffman es:

- Si la diferencia *DPCM* es positiva, se concatenan los *SSSS bits* que corresponden al valor de *DPCM*.

- Si la diferencia *DPCM* es negativa se concatenan los *SSSS bits* que representan el valor de la diferencia *DPCM* pero en complemento a dos.

Estos *SSSS bits adicionales* son anexados al primer bit de sus correspondientes códigos base. Estos códigos base son diferentes para la Luminancia y la Crominancia se pueden ver en la ***Tablas A.3*** y ***A.4*** en el ***anexo A***.

La ***Tabla 3.11*** es de gran ayuda al momento de codificar las diferencias *DPCM*, la columna etiquetada como *"Diferencia DPCM"* nos permite clasificar el valor de las diferencias en distintas categorías, una vez que se localiza la categoría correspondiente, la ***Tablas A.3*** y ***A.4***, proveen el código base al cual solo se le concatenan los *SSSS bits adicionales*.

SSSS	*Valores DPCM*	*Bits adicionales*
0	*0*	*-*
1	*-1 , 1*	*0 , 1*
2	*-3, -2 , 2, 3*	*00, 01 , 10, 11*
3	*-7....-4 , 4....7*	*000....011 , 100....111*
4	*-15....-8 , 8....15*	*0000....0111 , 1000....1111*
5	*-31....-16 , 16....31*	*00000....01111 , 10000....11111*
6	*-63....-32 , 32....63*	*000000....011111 , 100000....111111*
7	*-127.....-64 , 64.....127*	*0000000....0111111 , 1000000....1111111*
8	*-255....-128 , 128....255*	*00000000....01111111 , 10000000....11111111*
9	*-511.....-256 , 256.....511*	*000000000....011111111 , 100000000....111111111*
10	*-1023.....-512 , 512.....1023*	*0000000000....0111111111 , 1000000000....1111111111*
11	*-2047....-1024 , 1024....2047*	*00000000000....01111111111 , 10000000000....11111111111*
12	*- 4095.....-2048 , 2048.....4095*	*000000000000....011111111111 , 100000000000....111111111111*
13	*-8191.....-4096 , 4096.....8191*	*0000000000000....0111111111111 , 1000000000000....1111111111111*
14	*-16383.....-8192 , 8192.....16383*	*00000000000000....01111111111111 , 10000000000000....11111111111111*
15	*-32767.....-16384 , 16384.....32767*	*000000000000000....011111111111111 , 100000000000000....111111111111111*
16	*32768*	*-*

Tabla 3.11.- Categorías DPCM para codificación entrópica

Para realizar la decodificación de las diferencias *DPCM*, se sigue un procedimiento muy similar (el cual se describirá en la ***sección 4.3.5***) utilizando el mismo criterio de los códigos Huffman.

3.3.4.5.- Codificación de los coeficientes de AC

Los 63 coeficientes de *AC* restantes de cada *unidad de datos* se ordenan en un arreglo unidimensional mediante un ordenamiento conocido como secuencia *Zigzag*, al agrupar los coeficientes en este arreglo, los coeficientes de más baja frecuencia tienden a presentarse al inicio del arreglo y los coeficientes de alta frecuencia al final.

Usualmente los coeficientes de baja frecuencia son diferentes de cero y los de alta son cero, en la ***Figura 3.19*** el orden a seguir por el *Zigzag* es indicado por la dirección de las flechas, hay que señalar que aunque el coeficiente *DC* se codificada de diferente manera pertenece al arreglo unidimensional creado por el *Zigzag*, como primer elemento.

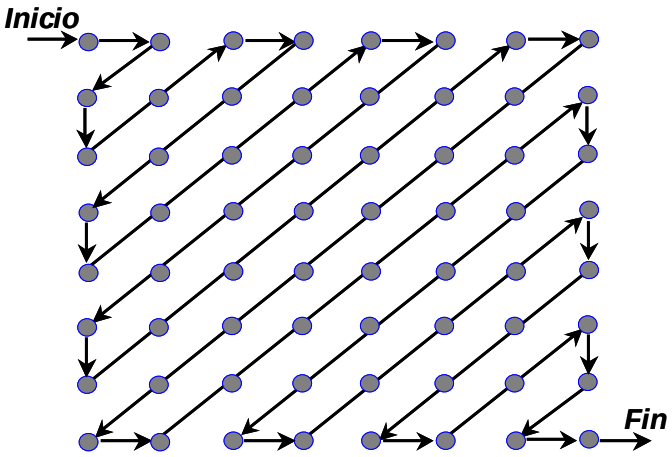


Figura 3.19.- Ordenamiento mediante la secuencia zigzag.

La secuencia *Zigzag* es una parte importante del modelo de codificación, el objetivo de esta secuencia es agrupar los coeficientes ordenándolos aproximadamente de acuerdo a la probabilidad decreciente de su ocurrencia. Como se muestra en la ***Figura 3.20*** se logra apreciar que conforme el índice de los elementos del arreglo unidimensional se incrementa las probabilidades de que se presente un coeficiente de valor no cero son demasiado pequeñas.

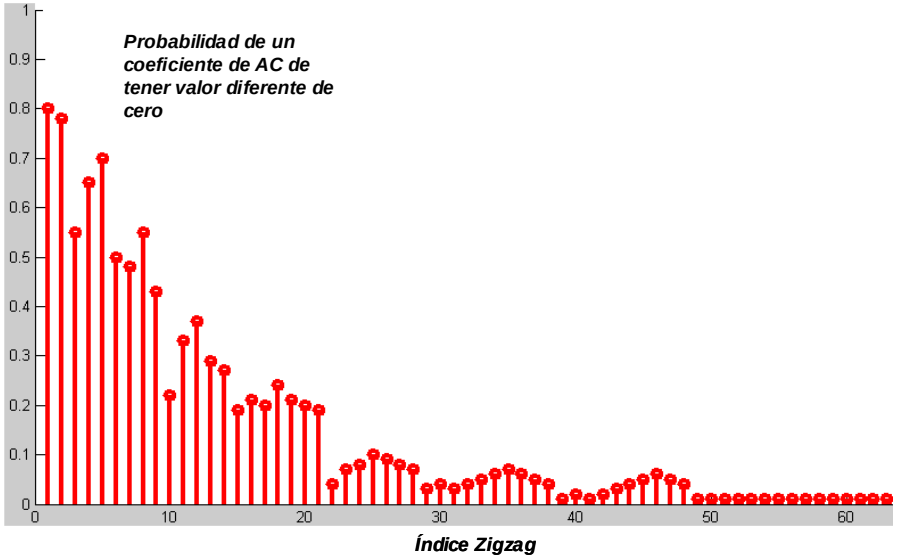


Figura 3.20.- Probabilidad de ocurrencia de los coeficientes *DCT* para el arreglo unidimensional resultado de la secuencia *zigzag.*

3.3.4.6.- Modelo estadístico para codificación de coeficientes de AC

Una vez que los coeficientes *AC* son agrupados de acuerdo al orden zigzag se procede a realizar la codificación entrópica, el proceso de cuantización provoca que muchos de los coeficientes *AC* se vuelvan cero (sobre todo los de alta frecuencia), este hecho es aprovechado por el codificador entrópico. El codificador Huffman podría asignar símbolos a los coeficientes cuyo valor sea cero, sin embargo está estrategia no es la más óptima [1][16].

Un mejor modelo para el codificador Huffman usa símbolos que combinan el número de coeficientes de valor cero, que anteceden a un coeficiente de valor diferente de cero. Los coeficientes de valor diferente de cero se agrupan en distintas categorías dependiendo de su magnitud, esas *categorías SSSS* se incrementan cuasi-logaritmicamente exactamente en la misma forma que las categorías *DPCM* y la estrategia de codificación por lo tanto es muy similar.

Consecuentemente, excepto para agrupaciones muy grandes de ceros, los códigos Huffman (también llamados códigos base) son asignadas a todas las combinaciones posibles de agrupaciones de ceros y de *categorías SSSS*. Estos códigos base son seguidos por *SSSS bits adicionales* que al concatenarse a los códigos base especifican completamente el signo y la magnitud de un coeficiente de valor diferente de cero. La ***Tabla 3.12***, muestra las *categorías SSSS* para los coeficientes de *AC,* así como los *SSSS bits adicionales* que corresponden a cada categoría.

SSSS	*Coeficientes AC*	*Bits adicionales*
1	*-1 , 1*	*0 , 1*
2	*-3, -2 , 2, 3*	*00, 01 , 10, 11*
3	*-7....-4 , 4....7*	*000....011 , 100....111*
4	*-15....-8 , 8....15*	*0000....0111 , 1000....1111*
5	*-31....-16 , 16....31*	*00000....01111 , 10000....11111*
6	*-63....-32 , 32....63*	*000000....011111 , 100000....111111*
7	*-127.....-64 , 64.....127*	*0000000....0111111 , 1000000....1111111*
8	*-255....-128 , 128....255*	*00000000....01111111 , 10000000....11111111*
9	*-511.....-256 , 256.....511*	*000000000....011111111 , 100000000....111111111*
10	*-1023.....-512 , 512.....1023*	*0000000000....0111111111 , 1000000000....1111111111*

Tabla 3.12.- Categorías magnitud para codificación de coeficientes *AC*

El número de coeficientes *AC* de valor cero que anteceden a un coeficiente de valor distinto a cero es designado como *RRRR*, la manera en que se agrupa a *RRRR* y las *categorías SSSS,* la provee la ***Tabla 3.13***.

RRRR \ SSSS	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	EOB	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
1	N/A	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
2	N/A	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
3	N/A	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
4	N/A	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
5	N/A	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
6	N/A	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
7	N/A	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
8	N/A	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
9	N/A	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
10	N/A	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
11	N/A	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
12	N/A	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
13	N/A	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
14	N/A	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
15	F/0 ZRL	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

Simbolos entropicos

No utilizados en el modo secuencial incluyendo el sistema base y entradas de ocho bits

Tabla 3.13.- Categorías RRRR y SSSS en valores hexadecimales

Las columnas señaladas en gris no son requeridas por el sistema básico, cuando se ha localizado la *categoría SSSS* a la cual pertenece un coeficiente y también se conoce el valor de *RRRR* para ese coeficiente, entonces ambos valores *RRRR/SSSS* son ubicados en la ***Tabla 3.13***, el valor que nos proporciona está tabla indica el código Huffman que se le asigna al coeficiente *AC*, adicionalmente para determinar la magnitud y el signo de dicho coeficiente son concatenados al código Huffman los *SSSS bits adicionales*.

Los códigos Huffman correspondientes a la *Tabla 3.13* se muestran el las ***Tablas A.5*** y ***A.6*** dadas en el ***anexo A***. Al igual que las tablas de códigos para las diferencias *DPCM*, los códigos de Huffman son definidos para la Luminancia y la Crominancia de manera distinta.

La columna *SSSS = 0* de la ***Tabla 3.13*** está reservada para dos símbolos especiales *EOB* y *ZRL*, *EOB* especifica la condición de fin de bloque y la longitud de cadena cero (*ZRL*) es usada para el caso raro, en el cual el número de coeficientes *AC* consecutivos de valor cero es mayor que 15, *ZRL* codifica una cadena de 15 ceros seguidos por un coeficiente de amplitud cero.

Note que la ***Tablas 3.11*** y ***3.12***, las cuales identifican las *categorías SSSS* para las diferencias *DPCM* y las *categorías SSSS* de los coeficientes *AC* respectivamente son idénticas excepto para el número de *categorías SSSS* requeridas. Esto no es un accidente, *MJPEG* trató de retener la mayoría de los elementos comunes para las diferentes partes del sistema.

3.3.4.7.- Ejemplo de codificación y decodificación de una unidad de datos

Para poder entender todos los procedimientos descritos en los apartados anteriores se procederá a realizar un ejemplo que ayude a aclarar y comprobar los conceptos citados. Para ilustrar este ejemplo se tomará la matriz que aparece en la ***Tabla 3.2***. El primer paso a seguir en la codificación es aplicar el corrimiento de nivel de –128 a la matriz original de la ***Tabla 3.2***, el resultado es la matriz desplazada que aparece en la ***Tabla 3.14***.

-44	*-44*	*-38*	*-36*	*-40*	*-62*	*-51*	*-42*
-74	*-72*	*-56*	*-50*	*-8*	*-4*	*-30*	*-62*
-73	*-72*	*-48*	*-8*	*0*	*-30*	*-54*	*-41*
-65	*-73*	*-8*	*12*	*117*	*-4*	*-62*	*-40*
-95	*-95*	*-83*	*-72*	*-72*	*-41*	*-4*	*-3*
-40	*-39*	*-38*	*-28*	*-4*	*-85*	*-82*	*-52*
-62	*-72*	*-46*	*-50*	*-95*	*0*	*-29*	*-61*
-64	*-52*	*-54*	*-50*	*-6*	*-7*	*-94*	*-93*

Tabla 3.14.- Matriz desplazada

A la matriz de la ***Tabla 3.14*** se le aplica la *DCT bidimensional*, está transformación genera la matriz de coeficientes desplazados dada por la ***Tabla 3.15***. La matriz de la ***Tabla 3.15*** difiere de la matriz de la ***Tabla 3.4***, solo en el valor del coeficiente *DCT* el cual es menor, todos los coeficientes *AC* son idénticos para la dos matrices (***Sección 3.3.4.1***) [1].

-353.75	*-57.48*	*-106.80*	*26.83*	*28.50*	*-22.82*	*17.18*	*16.74*
35.51	*-20.05*	*-13.05*	*-11.75*	*22.66*	*-0.28*	*-16.88*	*21.21*
-36.07	*53.30*	*25.87*	*19.96*	*-44.43*	*8.89*	*11.50*	*-30.60*
-33.64	*11.90*	*87.80*	*-42.04*	*-9.40*	*26.65*	*-18.71*	*-1.95*
12.50	*-12.85*	*-14.94*	*2.77*	*16.25*	*-23.86*	*0.50*	*17.45*
47.67	*27.36*	*-30.65*	*-12.82*	*15.01*	*6.18*	*13.15*	*3.77*
-0.21	*62.21*	*-11.50*	*-27.55*	*49.52*	*-27.73*	*-19.37*	*8.27*
-56.74	*-55.26*	*82.42*	*-20.57*	*-45.92*	*51.83*	*3.31*	*-44.08*

Tabla 3.15.- Matriz coeficientes desplazados

La matriz que aparece en la ***Tabla 3.15*** es la que debe ser cuantizada de acuerdo a la ***Tabla 3.9***, pues se considera que la matriz original pertenece a una imagen de Luminancia, además se considera que el valor del factor *DIV = 1,* lo que da como resultado los elementos de la matriz cuantizada (***Tabla 3.16***).

-22	*-5*	*-11*	*2*	*1*	*-1*	*0*	*0*
3	*-2*	*-1*	*-1*	*1*	*0*	*0*	*0*
-3	*4*	*2*	*1*	*-1*	*0*	*0*	*-1*
-2	*1*	*4*	*-1*	*0*	*0*	*0*	*0*
1	*-1*	*0*	*0*	*0*	*0*	*0*	*0*
2	*1*	*-1*	*0*	*0*	*0*	*0*	*0*
0	*1*	*0*	*0*	*0*	*0*	*0*	*0*
-1	*-1*	*1*	*0*	*0*	*1*	*0*	*0*

Tabla 3.16.- Matriz cuantizada

En la matriz de la ***Tabla 3.16*** se observa cómo mucho de sus elementos tienen valor cero (especialmente los que representan frecuencias altas), está matriz debe ser ordenada en un arreglo unidimensional de 64 elementos siguiendo el ordenamiento *Zigzag* (como se en la ***Figura 3.19***), el resultado de este ordenamiento es:

-22, -5, 3, -3, -2, -11, 2, -1, 4, -2, 1, 1, 2, -1, 1, -1, 1, 1, 4, -1, 2, 0, 1, 0, -1,-1,0, 0, 0, 0, 0, 0, 0, -1, 1, -1, -1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,0, 0, 0, 0, 1, 0, 0, 0, 0, 0.

El primer coeficiente que se debe codificar es el de *DC*, su valor es *-22* suponiendo que la matriz original es la primera *unidad de datos*, de una imagen entonces el valor *de DC* anterior es cero, por lo tanto según la ecuación **(3.17)**

$$DPCM = -22 - 0 = -22 \qquad \textbf{(3.17)}$$

Esta diferencia pertenece a la categoría 5 de la ***Tabla 3.11***, el código base que le corresponde es *110* según la ***Tabla A.3***, los bits adicionales son *01001*, que representan la magnitud de *-22*, estos son concatenados al código anterior, dando como resultado *11001001*.

Una vez codificada la diferencia *DPCM* se procede a codificar los coeficientes *AC*, el procedimiento es el siguiente:

El primer valor de *AC* es *-5*, al cual le corresponde la categoría 3 según la ***Tabla 3.12***, y es precedido por cero coeficientes de valor cero, por tal motivo le corresponde el valor *0/3* en la ***Tabla 3.13***, el código base proporcionado por la ***Tabla A.5*** es *100*, a este valor se le concatena la magnitud correspondiente a *-5* que es *010*, lo que genera que el primer coeficiente *AC* codificado sea *100010*; este procedimiento continua para todos los coeficientes *AC* restantes, los cuales se muestran en la ***Tabla 3.17***, esta tabla proporciona el valor del coeficiente *AC*, su *categoría SSSS* correspondiente, el número *RRRR* de ceros que le anteceden y por último su códigos Huffman.

Coeficiente	*RRRR*	*SSSS*	*RRRR/SSSS*	*Código*	*Bits adicionales*	*Coeficientes codificados*
-5	*0*	*3*	*0/3*	*100*	*10*	*100010*
3	*0*	*2*	*0/2*	*01*	*11*	*0111*
-3	*0*	*2*	*0/2*	*01*	*00*	*0100*
-2	*0*	*2*	*0/2*	*01*	*01*	*0101*
-11	*0*	*4*	*0/4*	*1011*	*0100*	*10110100*
2	*0*	*2*	*0/2*	*01*	*10*	*0110*
-1	*0*	*1*	*0/1*	*00*	*0*	*000*
4	*0*	*3*	*0/3*	*100*	*100*	*100100*
-2	*0*	*2*	*0/2*	*01*	*01*	*0101*
1	*0*	*1*	*0/1*	*00*	*1*	*001*
1	*0*	*1*	*0/1*	*00*	*1*	*001*
2	*0*	*2*	*0/2*	*01*	*10*	*0110*
-1	*0*	*1*	*0/1*	*00*	*0*	*000*
1	*0*	*1*	*0/1*	*00*	*1*	*001*

Tabla 3.17.- Coeficientes *AC* codificados

Coeficiente	*RRRR*	*SSSS*	*RRRR/SSSS*	*Código*	*Bits adicionales*	*Coeficientes codificados*
1	*0*	*1*	*0/1*	*00*	*1*	*001*
1	*0*	*1*	*0/1*	*00*	*1*	*001*
4	*0*	*3*	*0/3*	*100*	*100*	*100100*
-1	*0*	*1*	*0/1*	*00*	*0*	*000*
2	*0*	*2*	*0/2*	*01*	*10*	*0110*
1	*1*	*1*	*1/1*	*1100*	*1*	*11001*
-1	*1*	*1*	*1/1*	*1100*	*0*	*11000*
-1	*0*	*1*	*0/1*	*00*	*0*	*000*
-1	*7*	*1*	*7/1*	*11111010*	*0*	*111110100*
1	*0*	*1*	*0/1*	*00*	*1*	*001*
-1	*0*	*1*	*0/1*	*00*	*0*	*000*
-1	*0*	*1*	*0/1*	*00*	*0*	*000*
-1	*6*	*1*	*6/1*	*1111011*	*0*	*11110110*
1	*4*	*1*	*4/1*	*111011*	*1*	*1110111*
1	*9*	*1*	*9/1*	*111111001*	*1*	*1111110011*
EOB	*-*	*-*	*-*	*1010*	*-*	*1010*

Tabla 3.17.- Coeficientes *AC* codificados (continuación)

La secuencia final codificada queda de la siguiente manera:

1100100110001001110100010110110100011000010010001010010010110000001000001001100100000011011001110000001111101000010000001111011011101111111100111010.

El número total de bits requeridos para representar un bloque de *8x8* es solo *148*, en comparación con los *512* que se necesitan originalmente; por lo tanto se tiene un porcentaje de compresión de *(148/512) x 100% = 28.9%*. Esta compresión aún es baja, esto se debe a que en la etapa de cuantización no se eliminan muchos de los coeficientes *DCT,* de hecho casi la mitad de ellos son utilizados.

En el proceso de decodificación los valores que forman la secuencia *Zigzag* deben ser encontrados nuevamente, una vez que se obtienen, se aplica el procedimiento de *Zigzag inverso* obteniendo de nuevo la matriz de la ***Tabla 3.16***, los elementos de esta matriz se multiplican por los de la matriz de cuantización de luminancia, lo que da como resultado la matriz decuantizada que aparece en la ***Tabla 3.18*** .

-352	*-55*	*-110*	*32*	*24*	*-40*	*0*	*0*
36	*-24*	*-14*	*-19*	*26*	*0*	*0*	*0*
-42	*52*	*32*	*24*	*-40*	*0*	*0*	*-56*
-28	*17*	*88*	*-29*	*0*	*0*	*0*	*0*
18	*-22*	*0*	*0*	*0*	*0*	*0*	*0*
48	*35*	*-55*	*0*	*0*	*0*	*0*	*0*
0	*64*	*0*	*0*	*0*	*0*	*0*	*0*
-72	*-92*	*95*	*0*	*0*	*100*	*0*	*0*

Tabla 3.18.- Matriz decuantizada

Note que la matriz decuantizada contiene muchos elementos que presentan valor cero, aplicando la *IDCT* en dos dimensiones se obtiene la matriz de coeficientes inversos que se da en la ***Tabla 3.19***.

-44	*-34*	*-55*	*-32*	*-54*	*-35*	*-57*	*-43*
-87	*-49*	*-63*	*-37*	*0*	*-16*	*-65*	*-36*
-68	*-99*	*-53*	*-11*	*-7*	*-29*	*-28*	*-57*
-87	*-32*	*5*	*26*	*110*	*8*	*-69*	*-52*
-78	*-131*	*-90*	*-77*	*-61*	*-60*	*13*	*17*
-42	*-4*	*-24*	*-47*	*-3*	*-56*	*-101*	*-68*
-69	*-72*	*-68*	*-47*	*-51*	*-26*	*-37*	*-64*
-68	*-34*	*-67*	*-47*	*-31*	*-5*	*-74*	*-99*

Tabla 3.19.- Matriz de coeficientes inversos

Por último para obtener los valores finales solo se le suma la constante de *128* a cada uno de los elementos de la ***Tabla 3.19***, el resultado es la matriz reconstruida que aparece en la ***Tabla 3.20***.

84	*94*	*73*	*96*	*74*	*93*	*71*	*85*
41	*79*	*65*	*91*	*128*	*112*	*63*	*92*
60	*29*	*75*	*117*	*121*	*99*	*100*	*71*
41	*96*	*133*	*154*	*238*	*136*	*59*	*76*
50	*0*	*38*	*51*	*67*	*68*	*141*	*145*
86	*124*	*104*	*81*	*125*	*72*	*27*	*60*
59	*56*	*60*	*81*	*77*	*102*	*91*	*64*
60	*94*	*61*	*81*	*97*	*123*	*54*	*29*

Tabla 3.20.- Matriz reconstruida

Los valores de la matriz reconstruida difieren de los de la matriz original, esto se debe a las pérdidas sufridas en la etapa de cuantización, el efecto visual se puede apreciar en las siguientes figuras. La ***Figura 3.21*** muestra la representación visual de la matriz original, a su vez la ***Figura 3.22*** muestra la matriz reconstruida.

Figura 3.21.- Matriz Original

Figura 3.22.- Matriz reconstruida

RESUMEN

El objetivo de este tercer capitulo fue describir los conceptos principales que se toman en cuenta para la implementación del codificador-decodificador MJPEG sobre una secuencia de video utilizando un DSP. Se realizó una revisión de la implementación de la DCT así como de la IDCT, se

mostraron las formas en una y dos dimensiones de estas transformadas, así mismo se señaló la importancia de utilizar algoritmos rápidos para su implementación, pues la utilización de los criterios de la funciones base generan una gran cantidad de operaciones tanto para la DCT como para la IDCT.

Por otro lado en la ***sección 3.2*** *se mencionó que el estándar MJPEG tiene diferentes modos de operación, el modo a utilizar en este proyecto es el modo básico debido a que presenta un alto desempeño para una gran mayoría de aplicaciones.*

Además en la ***sección 3.3*** *se desglosó el modo de operación básico, se describieron sus bloques principales y la función que cada uno realiza en la codificación de una secuencia de video digital; para terminar el capítulo se mostró el proceso de codificación y decodificación sobre un bloque de 8x8, (unidad base en el modo básico). Este capítulo nos sirve para ilustrar la metodología que se debe seguir en la implementación del modo básico JPEG para una secuencia de imágenes digitales (video) sobre la arquitectura digital de un DSP.*

CAPITULO IV.- IMPLEMENTACIÓN DEL ESTÁNDAR MJPEG EN UN DSP

Este capítulo tiene como objetivo describir la implementación del sistema compresión – decompresión del estándar de vídeo digital MJPEG visto en el capitulo III. Este capítulo se divide en tres secciones, la primera de ellas presenta una breve descripción del sistema que se utiliza, así como las herramientas que nos permiten realizar su programación. La segunda sección describe como se realizó la etapa de compresión sobre el video almacenado en la memoria direccionada por el DSP. La última sección nos presenta todos los pasos que fueron necesarios para llevar a cabo el proceso de decompresión y así poder recuperar el video reconstruido.

4.1.- Procesador Digital de señales

DSP es acrónimo de *Digital Signal Processor*, nombre que describe la función de recibir señales, procesarlas, y entregar como resultado una nueva señal. Un *DSP* es un sistema basado en un microprocesador o microcontrolador que posee un juego de instrucciones y un Hardware optimizados para procesos que requieran operaciones numéricas a muy alta velocidad.

Debido a esto es especialmente útil para el procesado en tiempo real, necesitando de convertidores A/D y D/A en sus entradas y salidas respectivamente. Como todo sistema basado en un procesador programable necesitara una memoria donde almacenar los datos con los que trabajará y el programa que ejecutará [11] [20] [21].

El diseño de los *DPS´s* está enfocado a dos objetivos: al procesamiento de señales en tiempo real y al alto desempeño, debido a lo anterior, los *DSP´s* se pueden encontrar en todo tipo de aplicaciones portátiles que impliquen conversión de señales analógicas a digitales como Teléfonos celulares, Fáx, Modems, Control, etc.

4.1.1.- Arquitectura

Un *DSP* está diseñado teniendo en cuenta las tareas más habituales del procesado digital: sumas, multiplicaciones y retrasos, poseen una arquitectura tipo *Hardvard* y algunos pueden manejar números enteros o en punto flotante.

Las características básicas que componen a un *DSP* son [20]:

- Memoria de datos, memoria de programa y Acceso directo a memoria *(DMA)*
- Unidad de *multiplicación y acumulación (MAC)*
- Unidad aritmética lógica *(ALU)*
- Registros
- Periféricos

Así mismo el *DSP TMS320C6711* forma parte de la familia *TMS320C6000* de Texas Instruments (TI), la familia está diseñada para realizar cálculos numéricos intensivos por medio de la arquitectura tipo VLIW (Palabra de instrucción muy larga); consiste de varias unidades de ejecución corriendo en paralelo desarrollando múltiples instrucciones durante un simple ciclo de reloj. El procesador divide el trabajo dirigiendo cada operación a diferentes unidades de ejecución.

Dentro de la arquitectura del *6711* encontramos ocho unidades de ejecución, seis unidades lógicas aritméticas y dos unidades multiplicadoras, dichas unidades operan en paralelo y pueden realizar hasta ocho instrucciones de punto flotante durante un ciclo de reloj [10]. Los *DSP´s C6000* pueden realizar lecturas, escrituras y varias ejecuciones dentro de un mismo ciclo de reloj, gracias a que cuenta con buses separados para datos, programa y acceso directo a memoria.

4.1.2.- Diseño e implementación del sistema

El *DSP TMS320C6711* utilizado en este trabajo, está contenido en una tarjeta llamada *DSK*, la cual provee las herramientas de soporte necesarias tanto en Hardware como en Software, para realizar procesamiento en tiempo real. La ***Figura 4.1*** muestra un diagrama a bloques general de está tarjeta.

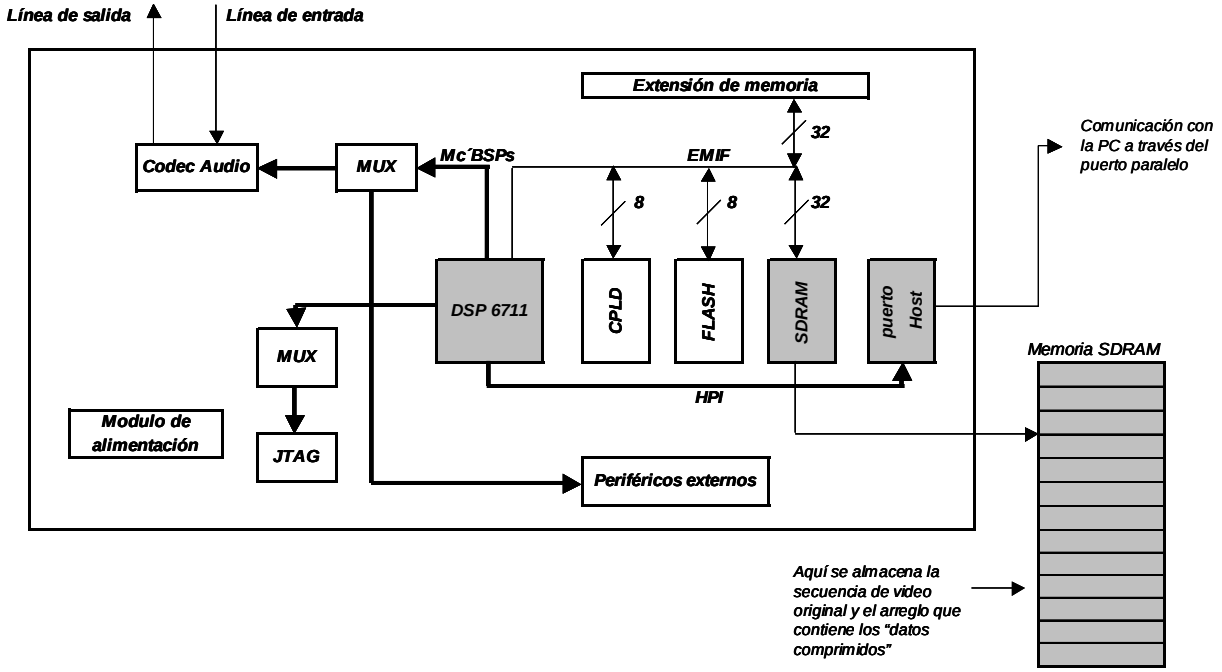


Figura 4.1.- Diagrama a bloques del DSK TMS320C6711

Está tarjeta además de contar con el *DSP* 6711, incluye un Codec de Audio que provee conversión A/D y D/A a una tasa de 8 KHz, también incluye 16 Mbytes de Memoria de Acceso Aleatorio Dinámica Síncrona *(SDRAM)*, 256 Kbytes de memoria Flash, además de contar con un puerto HOST *(HPI)*, el cual nos permite comunicarnos con la PC a través de un protocolo preestablecido [20] [21].

Cuenta con una interfaz *JTAG*, además nos provee una interfaz de memoria externa, que nos permite conectar dispositivos externos. Todo este sistema cuenta con los dispositivos necesarios para su alimentación, operando a una frecuencia de reloj de 150 MHz.

Como se mencionó el objetivo de los *DSP´s* es procesar señales en tiempo real, sin embargo, este trabajo no cumple con este criterio, pues las secuencias de video que se procesan son almacenadas en la memoria *SDRAM*, las cuales provienen de una PC. Estas secuencias han sido adquiridas previamente, su descarga en el *DSP* se realiza gracias a la ayuda del ambiente Code Composer Studio *(CCS),* (que es la interfaz gráfica que nos permite realizar la programación de la tarjeta); con ayuda del programa computacional Matlab se genera un archivo unidimensional que pueda ser reconocido por dicho ambiente *CCS*, el cual representa la información contenida en los cuadros de la secuencia, así mismo todas las imágenes de la secuencia se deben trasladar al espacio *YCrCb,* y gracias a la comunicación que se efectúa con el *HPI* es posible descargarlas en la tarjeta.

Tomando en cuenta estos criterios, los únicos elementos utilizados de la ***Figura 4.1*** para la realización de este trabajo, son los que se encuentran sombreados; no obstante y a pesar de que el sistema desarrollado no trabaja en tiempo real, su objetivo final es la implementación en dicho modo.

Esto se puede lograr con la ayuda de otra *tarjeta auxiliar,* que esté adquiriendo las muestras de video y el *DSP* pueda utilizar con ayuda del acceso directo de memoria mejorado *(EDMA)* – el cual es un controlador que transfiere datos entre el mapa de memoria y el CPU sin intervención de este último –.

Una manera de lograr una implementación en tiempo real, es utilizar una tarjeta auxiliar de adquisición de video; en la ***Figura 4.2*** se muestra un diagrama a bloques de dicha tarjeta.

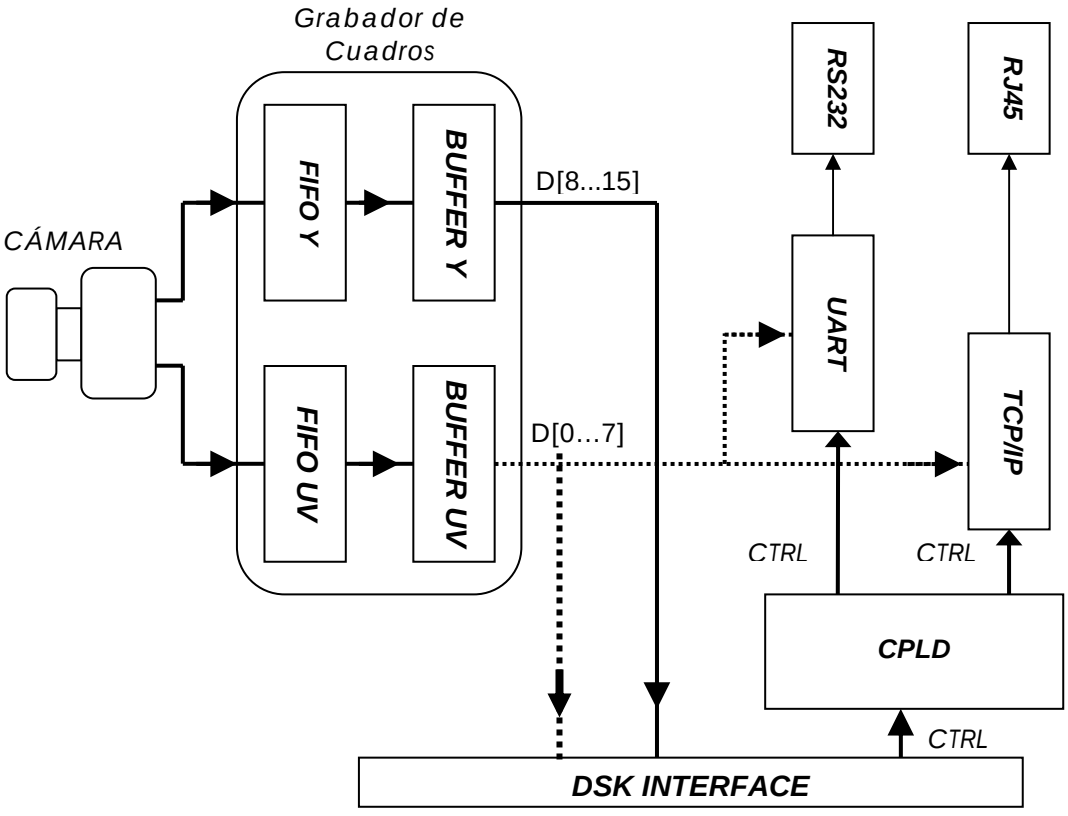


Figura 4.2.- Diagrama a bloques de la tarjeta para captura de video

Mediante una programación adecuada de los registros de la tarjeta de adquisición, se pueden generar formatos de imágenes cuya resolución es 720 píxeles x 480 líneas, en *YCrCb* o *RGB* crudo, la taza de actualización de cuadros puede ser programada para un valor constante de 30 cuadros/s o menos.

Los bloques FIFO (primero en entrar, primero en salir) del buffer de cuadro, son lo suficientemente grandes para guardar un cuadro completo, la interfaz del Hardware de la cámara a los bloques FIFO permite una actualización continua de los cuadros sin la intervención del *DSP.*

El *DSP* puede utilizar los datos provenientes de la *tarjeta de adquisición*, gracias a que cuenta con el controlador de acceso directo a memoria (DMA), el cual permite transferir datos de la memoria interna a los dispositivos externos sin la intervención del CPU. Los 16 canales de DMA mejorado (EDMA) pueden ser configurados independientemente para la transferencia de datos.

El EDMA puede manejar longitudes de datos de 8, 16 y 32 bits, los cuales pueden transferirse a la misma frecuencia que utiliza el CPU (150 MHz). La transferencia del EDMA puede dispararse por interrupciones de periféricos – Puertos Serial Buffereado Multicanal (McBSP), Timer o Interfaz externa de memoria (EMIF) –, o bien a través de pines externos [24][25].

De esta manera un esbozo del sistema en tiempo real consistirá en una captura de los cuadros por medio de la *tarjeta auxiliar de adqusición*, su correspondiente transferencia por EDMA hacía la *SDRAM*; una vez que él buffer de la *SDRAM* se llene, se deberá aplicar el algoritmo de compresión *MJPEG*, los datos comprimidos resultantes podrán ser enviados a la red utilizando el protocolo TCP/IP por medio del controlador que se encuentra en la *tarjeta auxiliar*.

Para realizar la decodificación se puede utilizar nuevamente el sistema *DSP-Tarjeta auxiliar de adquisición,* o bien utilizar una PC que nos permita recibir los datos comprimidos e implementar el algoritmo de decompresión *MJPEG*, asímismo se aprovechará el monitor de la PC para poder desplegar dicha secuencia de video.

Otra opción que nos permitiría realizar la implementación en tiempo real es utilizar algún dispositivo más avanzado de la familia *TMS320DM6XXX*, siendo este un *DSP* que cuenta con puertos de entrada de video *(VIP)* y puertos de salida de video *(VOP)*, en este caso, basta con direccionar el puerto correspondiente obteniendo las muestras de entrada y/o salida de video [24].

Antes de adentrarnos en la implementación de tiempo real, se optó por el diseño del Codificador-Decodificador *MJPEG* de una secuencia de video, utilizando solo el *DSP* y la *SDRAM*, los parámetros principales que se evaluaron fueron, factores de compresión, calidad visual y tiempos de ejecución, siendo este último el más importante pues es un indicativo de lo factible que puede llegar a ser una futura implementación en tiempo real; así mismo sus resultados nos indicarán como debe de ser mejorado el sistema. La implementación de los algoritmos de compresión y decompresión *MJPEG* fue utilizando lenguaje *C*, esto es posible gracias al compilador que nos provee el ambiente *CCS.*

4.2.- Algoritmo para la implementación de la compresión MJPEG

Considerando que la secuencia de video es almacenada en una sección de la memoria *SDRAM* de 32 bits, con lo cual se está simulando la adquisición y almacenamiento en un buffer para su procesamiento, tal como se muestra en la ***Figura 4.3***.

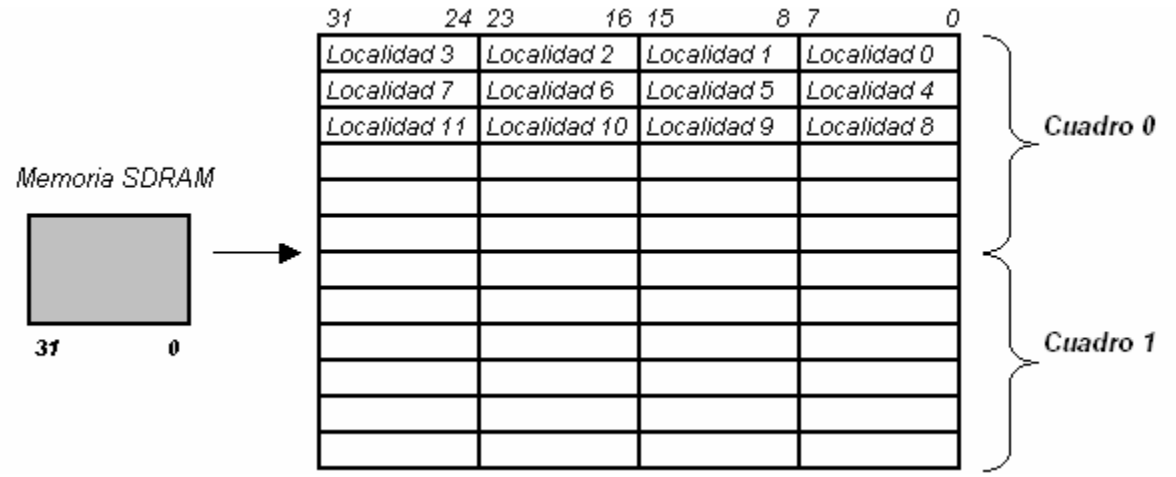


Figura 4.3.- Almacenamiento de la secuencia de video digital en memoria del *DSP*

La secuencia de video almacenada en la *SDRAM* debe seguir el orden lineal. Debido a que se utiliza el formato *QCIF* (176 píxeles x 144 líneas) con una profundidad de ocho bits por componente de color, cada píxel se almacena en una localidad de memoria de ocho bits. Como el ancho de palabra de la *SDRAM*, es de 32 bits es posible almacenar en cada localidad, cuatro localidades de ocho bits, tal como se muestra en la ***Figura 4.3***.

Si la secuencia de video a procesar está formada por tres componentes, se almacenarán en tres secciones diferentes, el tamaño de cada sección será el mismo y estará dado por *176 x 144 x número de cuadros*. Después se declara un arreglo tridimensional con el cual se pueda hacer uso de la información que se encuentra contenida en la *SDRAM*, gracias al archivo de comandos '.cmd' y a la directiva PRAGMA del lenguaje *C* es posible apartar una sección de memoria; su declaración es:

```
#pragma DATA_SECTION (original_y,"mia");
```

El nombre *"mia"* indica al compilador la sección de memoria que se está apartando en el archivo '.cmd', la asignación *original_y* indica el nombre de la variable que se mapeara en dicha sección de memoria. *original_y* es el arreglo que contiene los cuadros de la secuencia de video que pertenecen a la componente de luminancia.

Una vez que se tiene almacenada la secuencia de video, se procede a aplicar el algoritmo general de compresión *MJPEG*, que se muestra en la ***Figura 4.4***.

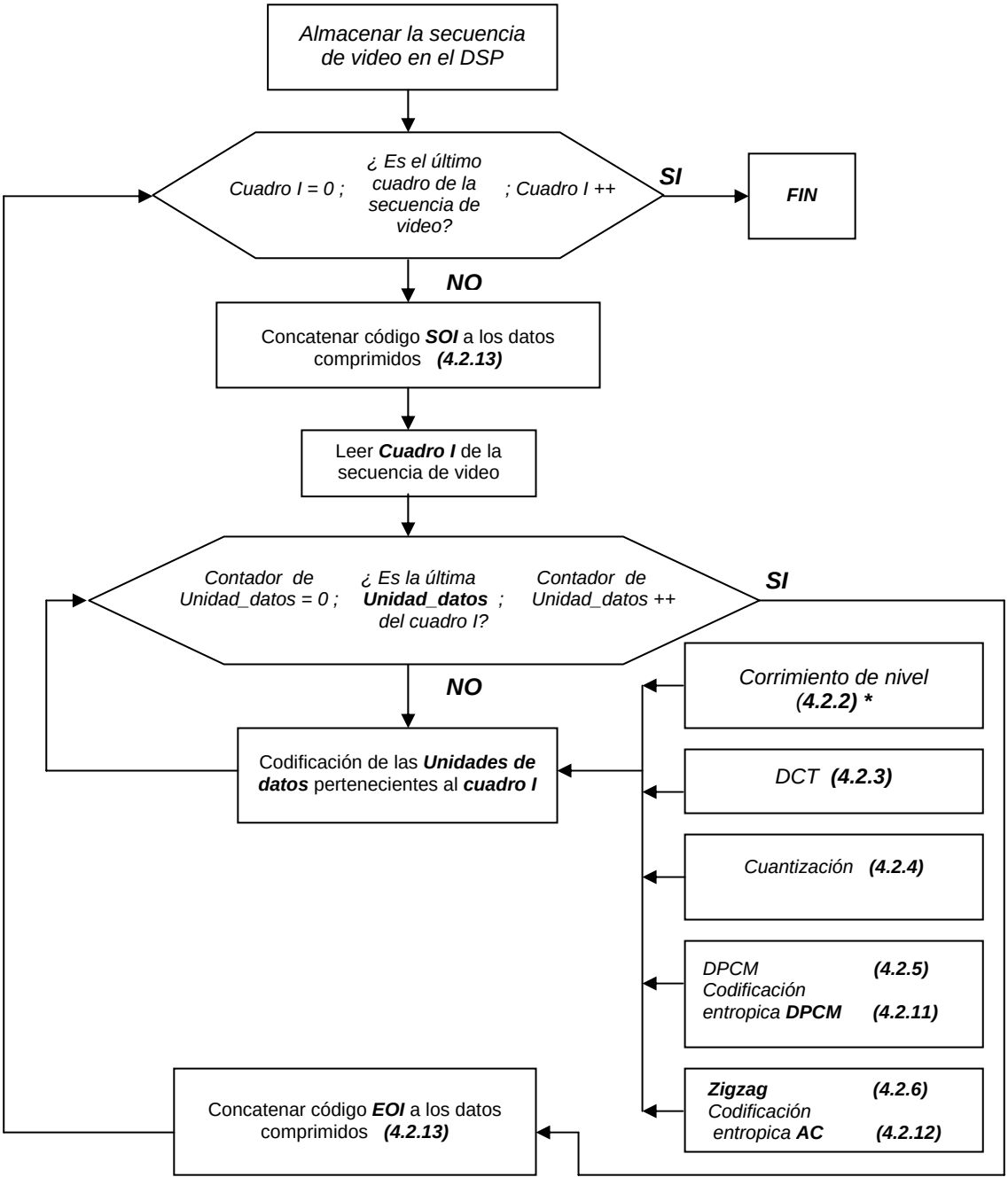


Figura 4.4.- Diagrama de flujo del algoritmo general de Compresión MJPEG utilizado en la implementación del sistema

Antes de iniciar el proceso de codificación se concatena el código *SOI* (Inicio de cuadro) al inicio del arreglo que almacenará los datos comprimidos, inmediatamente después se entra en un ciclo en el cual se procesan todas las *unidades de datos (8 x 8)* del cuadro I, este proceso consta de las etapas de corrimiento de nivel, *DCT*, cuantización, codificación entrópica *DPCM* y *AC (**Sección 3.3.4**)*.

* Hace referencia a la sección de este capítulo donde se describe el procedimiento para su implementación.

Una vez que se han procesado todas las unidades de datos se concatena el código *EOI* (Fin de cuadro) a los datos comprimidos, el proceso se repite nuevamente hasta que todos los cuadros de la secuencia finalizan.

4.2.1.- Lectura de los cuadros y las unidades de datos

El primer proceso en la realización de la codificación de la secuencia de video es la lectura de los cuadros. En vista que la unidad fundamental de procesamiento es la *unidad de datos*, lo primero que se debe de hacer es realizar la lectura de dichas unidades, esto se logra a través del código general:

```
n = 0;
  do {
    m = 0;
     do {

         for ( j = 0 ; j < 8 ; j + + )
           {
          for ( i = 0 ; i < 8 ; i + + )
            unidad_datos [ j ] [ i ] = original_y [ cuadro_I ] [ j + n ] [ i + m ];
          }
       --- Procesos de compresión (DCT,cuantización,....)
     m = m + 8;
     }while ( m < 176 );

   n = n + 8 ;}
  while ( n < 144 );
```

Debido a que los datos están almacenados de manera lineal y las *unidades de datos* son estructuras bidimensionales, se utilizan dos ciclos *For* anidados los cuales permiten reagrupar la información, a su vez las variables *m* y *n* junto con los ciclos *While* permiten realizar el barrido de todas las *unidades de datos* de la imagen que se está procesando (el código total para la implementación de la etapa de compresión se puede ver en el ***Anexo D***).

El incrementar el valor de la variable *m* en ocho es equivalente a leer la siguiente *unidad de datos* en la dirección horizontal; incrementar el valor de la variable *n* en ocho es equivalente a leer la siguiente *unidad de datos* en la dirección vertical.

De esta manera las variables *m* y *n* nos permiten realizar un barrido secuencial de todas las unidades de datos (***Sección 3.2.1.1***), pertenecientes a un mismo cuadro, este proceso se repetirá para todos los cuadros de la secuencia de video. Un ejemplo de este proceso se observa en la ***Figura 4.5***.

En la ***Figura 4.5***, se muestra la codificación de todas las *unidades de datos* que forman dos cuadros de la secuencia de video, así como la ubicación en la *SDRAM* de los elementos que forman las primeras *unidades de datos* de cada cuadro.

Las localidades de memoria que forman a la *unidad de datos 0* del primer cuadro, se indican en la ***Tabla 4.1***, en dicha tabla se observa como las localidades del primer renglón van desde la 0 hasta la 7, para el segundo renglón sus localidades comienzan desde la 176 hasta la 183, cada renglón posterior comenzará con un valor 176 veces mayor que el valor inicial del renglón anterior.

Para poder procesar el siguiente cuadro de la secuencia es necesario modificar el valor de la variable *cuadro I*, incrementar el valor de esta variable en uno es equivalente a iniciar el direccionamiento en las 25,344 localidades posteriores al valor inicial del cuadro anterior. Utilizando nuevamente el barrido secuencial, observamos que la *unidad de datos 0* del segundo cuadro, está formada por los elementos que aparecen en la ***Tabla 4.2.***

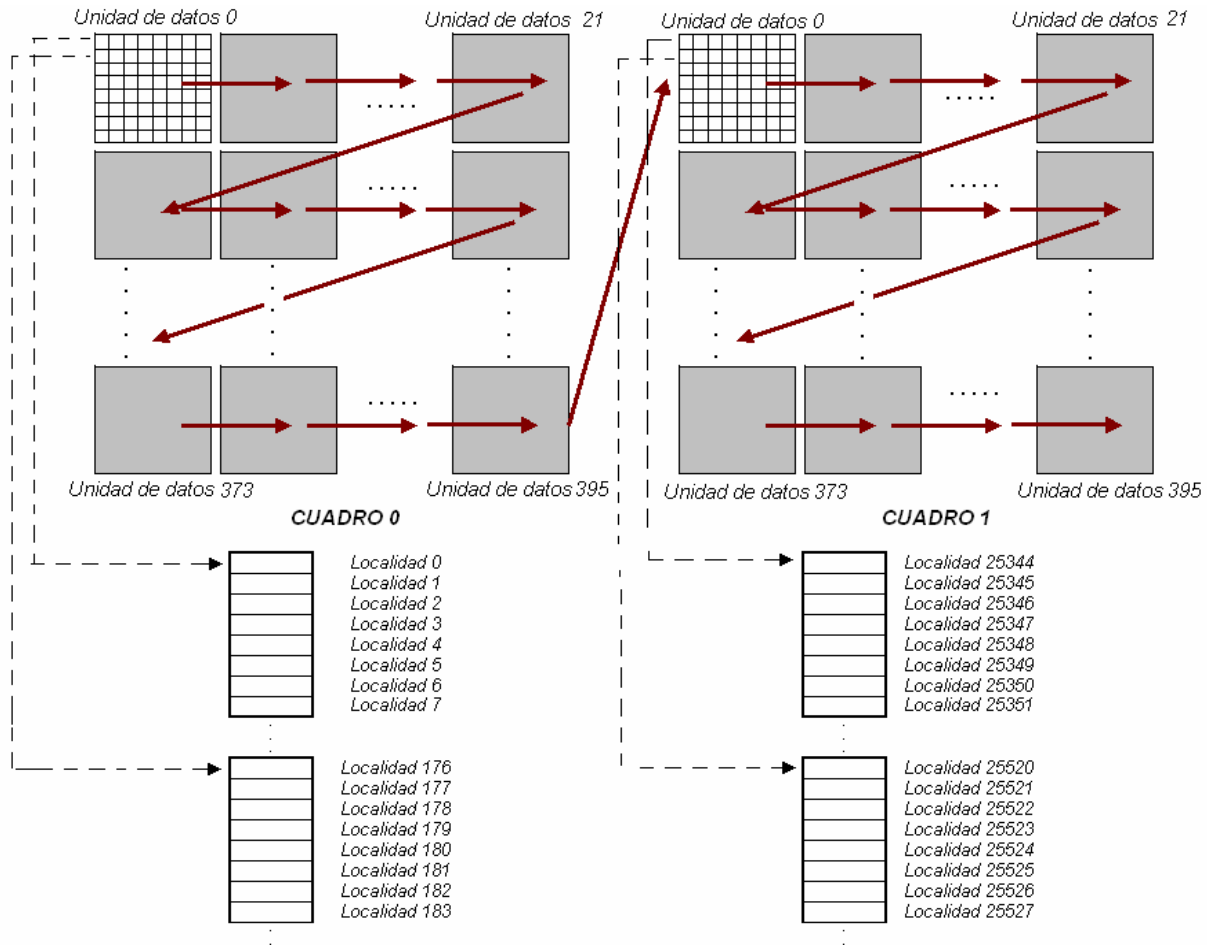


Figura 4.5.- Lectura secuencial de las unidades de datos para los primeros dos cuadros de la secuencia de video

Localidad 0	*Localidad 1*	*Localidad 2*	*Localidad 3*	*Localidad 4*	*Localidad 5*	*Localidad 6*	*Localidad 7*
Localidad 176	*Localidad 177*	*Localidad 178*	*Localidad 179*	*Localidad 180*	*Localidad 181*	*Localidad 182*	*Localidad 183*
Localidad 352	*Localidad 353*	*Localidad 354*	*Localidad 355*	*Localidad 356*	*Localidad 357*	*Localidad 358*	*Localidad 359*
Localidad 528	*Localidad 529*	*Localidad 530*	*Localidad 531*	*Localidad 532*	*Localidad 533*	*Localidad 534*	*Localidad 535*
Localidad 704	*Localidad 705*	*Localidad 706*	*Localidad 707*	*Localidad 708*	*Localidad 709*	*Localidad 710*	*Localidad 711*
Localidad 880	*Localidad 881*	*Localidad 882*	*Localidad 883*	*Localidad 884*	*Localidad 885*	*Localidad 886*	*Localidad 887*
Localidad 1056	*Localidad 1057*	*Localidad 1058*	*Localidad 1059*	*Localidad 1060*	*Localidad 1061*	*Localidad 1062*	*Localidad 1063*
Localidad 1232	*Localidad 1233*	*Localidad 1234*	*Localidad 1235*	*Localidad 1236*	*Localidad 1237*	*Localidad 1238*	*Localidad 1239*

Tabla 4.1.- *Unidad de datos 0* del primer cuadro de la secuencia.

Localidad 25344	*Localidad 25345*	*Localidad 25346*	*Localidad 25347*	*Localidad 25348*	*Localidad 25349*	*Localidad 25350*	*Localidad 25351*
Localidad 25520	*Localidad 25521*	*Localidad 25522*	*Localidad 25523*	*Localidad 25524*	*Localidad 25525*	*Localidad 25526*	*Localidad 25527*
Localidad 25696	*Localidad 25697*	*Localidad 25698*	*Localidad 25699*	*Localidad 25700*	*Localidad 25701*	*Localidad 25702*	*Localidad 25703*
Localidad 25872	*Localidad 25873*	*Localidad 25874*	*Localidad 25875*	*Localidad 25876*	*Localidad 25877*	*Localidad 25878*	*Localidad 25879*
Localidad 26048	*Localidad 26049*	*Localidad 26050*	*Localidad 26051*	*Localidad 26052*	*Localidad 26053*	*Localidad 26054*	*Localidad 26055*
Localidad 26224	*Localidad 26225*	*Localidad 26226*	*Localidad 26227*	*Localidad 26228*	*Localidad 26229*	*Localidad 26230*	*Localidad 26231*
Localidad 26400	*Localidad 26401*	*Localidad 26402*	*Localidad 26403*	*Localidad 26404*	*Localidad 26405*	*Localidad 26406*	*Localidad 26407*
Localidad 26576	*Localidad 26577*	*Localidad 26578*	*Localidad 26579*	*Localidad 26580*	*Localidad 26581*	*Localidad 26582*	*Localidad 26583*

Tabla 4.2.- *Unidad de datos 0* del segundo cuadro de la secuencia.

Este barrido secuencial se ha descrito para una secuencia que posee solamente una componente, para procesar las tres componentes se realiza un solo barrido – es decir se utiliza la codificación entrelazada de las *unidades de datos (**sección 3.3.3**)* –.

Una vez que se realiza la lectura de determinada *unidad de datos* se le deben aplicar los procesos correspondientes de compresión, y una vez que estos terminan se debe realizar la lectura de las siguientes *unidades de datos* y tambien aplicar sus respectivos procesos de compresión; lo que da como resultado que todos los datos comprimidos pertenecientes a las tres componentes se ubiquen en un solo arreglo.

4.2.2.- Etapa de corrimiento

La primera etapa en la codificación es el corrimiento de nivel, dicho proceso se describió en la ***Sección 3.3.4.1***, su implementación es bastante fácil, solo se resta el valor de –128 de cada elemento de la unidad de datos y el resultado de esta operación se almacena en la variable *unidad_d_c[8][8]*.

4.2.3.- Etapa DCT

Como se vio en la ***Sección 3.3.4.2***, la etapa *DCT* es la siguiente en el proceso de codificación, para la realización de la *DCT bidimensional* sobre una *unidad de datos* se optó por utilizar el algoritmo unidimensional optimizado Wang-Suehiro-Hatori (***Sección 3.1.6***) para reducir al máximo el número de operaciones [4][1].

La *DCT bidimensional*, es aplicada por una subrutina que es llamada cada vez que se desea ejecutar este proceso; primero se aplica la *DCT unidimensional* sobre cada renglón del arreglo *unidad_d_c[8][8]*, con lo cual se genera la matriz intermedia, después se aplica la *DCT*

unidimensional en cada columna de esta nueva matriz *intermedia[8][8]*, el resultado es el arreglo transformado llamado *coeficientes[8][8].*

4.2.4.- Etapa de cuantización

Como se observo en la ***Sección 3.3.4.2*** una vez que se ha transformado la *unidad de datos,* ésta debe ser cuantizada de acuerdo a sus respectiva tablas de cuantización, esto se logra por medio de la ecuación **(3.16)**. Donde el *Factor DIV* permite escalar los valores de cada coeficiente en las tablas de cuantización, las cantidades que se le asignan a este factor para este trabajo, son ***2, 1, 0.5, 0.3*** y ***0.2***.

La definición de la ecuación **(3.16)** implica el uso de operaciones de división, que generalmente son más costosas computacionalmente, con el fin de optimizar procesos, se decidió utilizar operaciones de multiplicación en lugar de las divisiones, para lograrlo se utilizan tablas de cuantización cuyos elementos son el inverso matemático de los que les corresponden a las ***Tablas 3.9*** y ***3.10*** originales; las ***Tablas A.1*** y ***A.2*** son las que se utilizarán en la etapa de codificación, las cuales se citan en los anexos.

El resultado de la operación de multiplicación genera un resultado fraccionario, el cual debe ser transformado a un valor entero signado, este resultado nuevamente se almacena en el arreglo *unidad_d_c[8][8]*. Para efectuar la cuantización de las *unidades de datos* de Crominancia, simplemente hay que realizar la multiplicación por los elementos de la tabla de Crominancia.

4.2.5.- Obtención de las diferencias DPCM

Como se mencionó en la ***sección 3.3.4.3*** los coeficientes *DC* se preparan separadamente de los de *AC*, antes de que la codificación entrópica se aplique es necesario obtener las diferencias correspondientes entre los coeficientes *DC* del bloque que se está procesando y el coeficiente *DC* del bloque inmediato que le precede.

Las líneas de código que realizan la operación anterior son:

```
zig [0 ] = unidad_d_c[0] [0] – yny ;
yny = unidad_d_c [0] [0] ;
```

La primera vez que se ejecuta dicha función, la variable *yny* debe ser inicializada a cero y en las siguientes ejecuciones su valor debe conservarse, el valor *DPCM* se obtiene de la resta del primer elemento del arreglo *unidad_d_c [8][8]* con la variable *yny,* almacenando el resultado en el primer elemento del arreglo z*ig [64]* [13].

4.2.6.- Ordenamiento Zigzag

El ordenamiento *Zigzag* es la etapa previa a la codificación entrópica de los coeficientes de *AC*, la implementación de está etapa tiene como objetivo la creación de un arreglo unidimensional *zig [64]* que está basado en los elementos del arreglo *unidad_d_c [8][8],* siguiendo el ordenamiento *Zigzag (**Sección 3.3.4.5**).* Para realizar este ordenamiento se crea un algoritmo que se divide en tres etapas, basándose en el hecho de que el ordenamiento tiene como partes principales:

- *Incrementos de una unidad* : Que equivale a desplazar un elemento de izquierda a derecha en la dirección horizontal y cero elementos en la dirección vertical.
- *Incrementos de siete unidades* : Que equivale a desplazarse un elemento de derecha a izquierda en la dirección horizontal y un elemento de arriba hacia abajo en la dirección vertical.
- *Incremento de ocho elementos:* Equivale a desplazarse cero elementos en la dirección horizontal y un elemento de arriba hacia abajo en la dirección vertical.
- *Decremento de siete elementos:* Equivale a desplazarse un elemento de izquierda a derecha en la dirección horizontal y un elemento de abajo hacia arriba en la dirección vertical.

La primera etapa tiene como objetivo realizar el agrupamiento que aparece en la ***Figura 4.6***, para lograr este ordenamiento, los valores del arreglo *unidad_d_c* se deben incrementar de acuerdo a la siguiente secuencia: *1,+7,+8,-7,-7,+1,+7,+7,+7,+8,-7,-7,-7,-7,+1,+7,+7,+7,+7,+7,+8,-7,-7,-7,-7,-7,-7*.

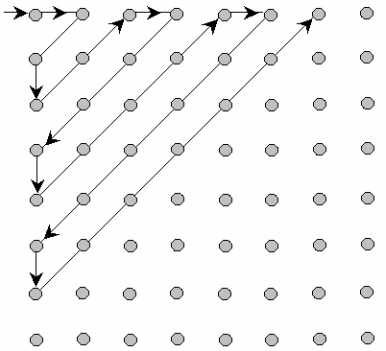


Figura 4.6.- Ordenamiento Zig-Zag primera etapa.

La segunda etapa tiene como objetivo cumplir con el agrupamiento que aparece en la ***Figura 4.7***, esto se logra incrementando los valores del arreglo *unidad_d_c* por medio de la secuencia *1,+7,+7,+7,+7,+7,+7,+7,+7*.

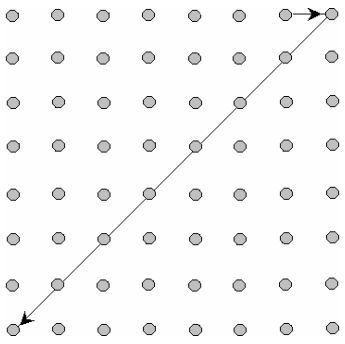


Figura 4.7.- Ordenamiento Zig-Zag segunda etapa.

La tercera y última etapa tiene como función cumplir con el agrupamiento que aparece en la ***Figura 4.8***, para realizarlo se debe incrementar el arreglo *unidad_d_c* de acuerdo a la secuencia *1,-7,-7,-7,-7,-7,-7,+8,+7,+7,+7,+7,+7,+1,-7,-7,-7,-7,+8,+7,+7,+7,+1,-7,-7,+8,+7,+1*.

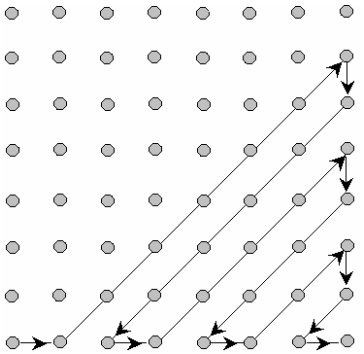


Figura 4.8.- Ordenamiento *Zigzag* tercera etapa.

Los algoritmos que nos permiten realizar estas tres etapas se muestran en las ***Figuras 4.9***, ***4.10*** y ***4.11***.

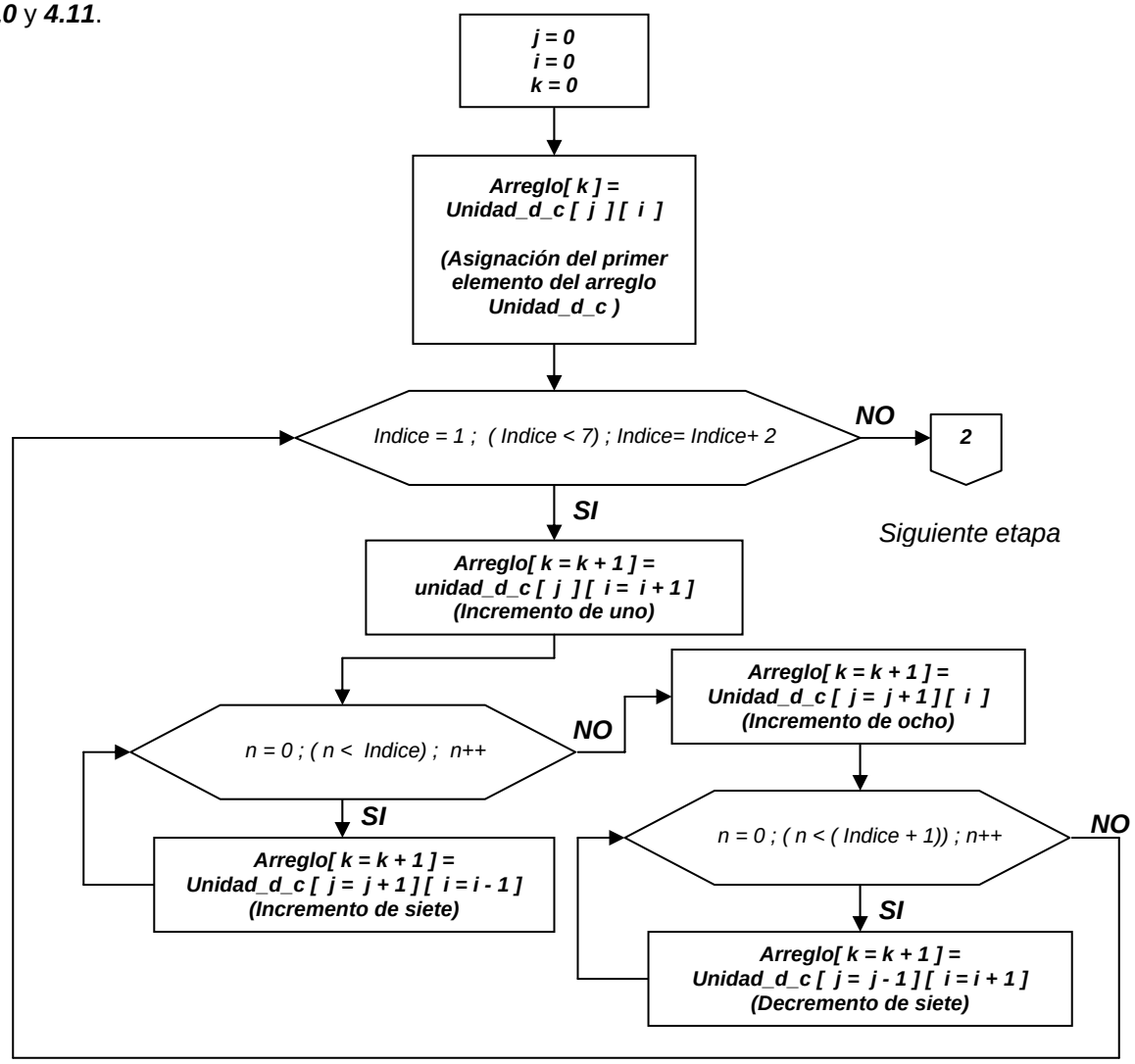


Figura 4.9.- Algoritmo ordenamiento *Zigzag* primera etapa.

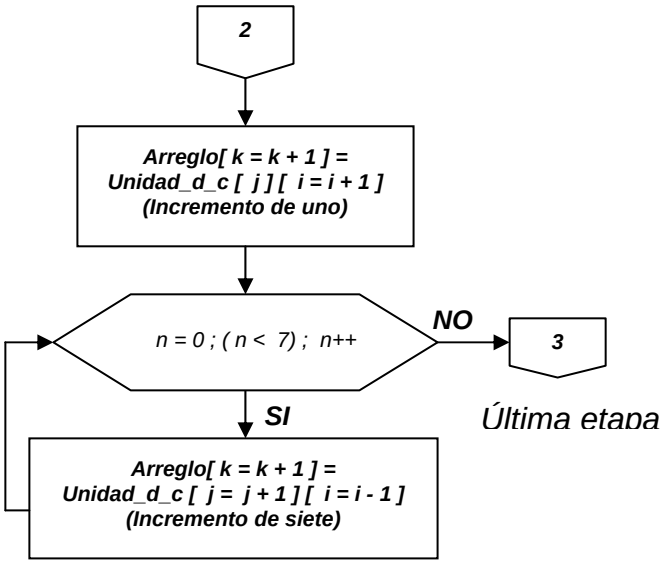


Figura 4.10.- Algoritmo ordenamiento *Zigzag* segunda etapa

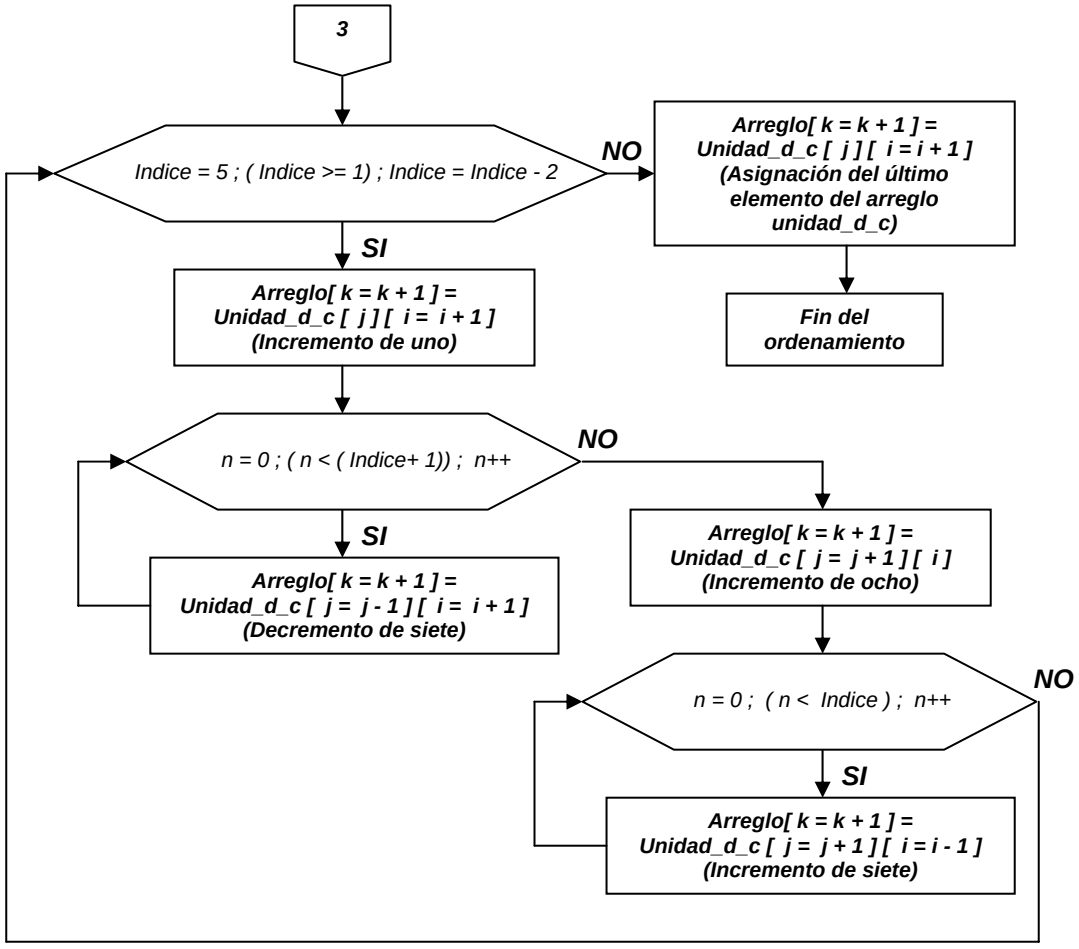


Figura 4.11.- Algoritmo ordenamiento *Zigzag* tercera etapa

Siguiendo estas tres etapas los elementos de la *unidad_d_c* (que contiene el valor *DPCM* y los coeficientes *AC*) quedan bajo el ordenamiento *Zigzag* [13].

4.2.7.- Rutina de concatenación

Una vez que se encuentra el valor *DPCM* y se ordenan los elementos en el arreglo *zig*, se debe de aplicar la codificación entrópica que es la última etapa de la compresión, para poder realizar esto se utilizan los códigos Huffman (***Tablas A3 - A6*** anexos), como se vio en el ejemplo de la ***Sección 3.3.4.7***.

Para realizar está implementación en la memoria direccionada por el *DSP*, es necesario crear una rutina especial de concatenación, la cual se basa en el algoritmo de la ***Figura 4.12***, esta rutina surge del hecho de que la memoria del *DSP* solo se puede manejar valores de 8, 16 o 32 bits de ancho de palabra y los códigos Huffman además de ser variables no cumplen con estas longitudes [6] [13] [8].

Para la realización de está rutina, se tomaron en cuenta dos criterios, primero el código Huffman y segundo su longitud, es decir, el número de bits que lo forman. Para facilitar la utilización de los códigos (tanto de *DC* como de *AC*) se optó por manejarlos en su representación decimal; por ejemplo los códigos Huffman de *DC* se muestran en la ***Tabla 4.3*** y algunos de los códigos de *AC* en la ***Tabla 4.4***.

Categoria SSSS	*Código Huffman*	*Longitud*	*Representación decimal*
0	*00*	*2*	*0*
1	*010*	*3*	*2*
2	*011*	*3*	*3*
3	*100*	*3*	*4*
4	*101*	*3*	*5*

Tabla 4.3.- Representación decimal de los códigos Huffman DC de Luminancia

RRRR/SSSS	*Código Huffman*	*Longitud*	*Representación Decimal*
0/0	*1010*	*4*	*10*
0/1	*00*	*2*	*0*
0/2	*01*	*2*	*1*
.	.	.	.
.	.	.	.
.	.	.	.
1/1	*1100*	*4*	*12*
1/2	*11011*	*5*	*27*
1/3	*1111001*	*7*	*121*
.	.	.	.
.	.	.	.
.	.	.	.
2/1	*11100*	*5*	*28*
2/2	*11111001*	*8*	*249*
2/3	*1111110111*	*10*	*1015*
.	.	.	.
.	.	.	.
.	.	.	.

Tabla 4.4.- Representación decimal de los códigos Huffman AC de Luminancia

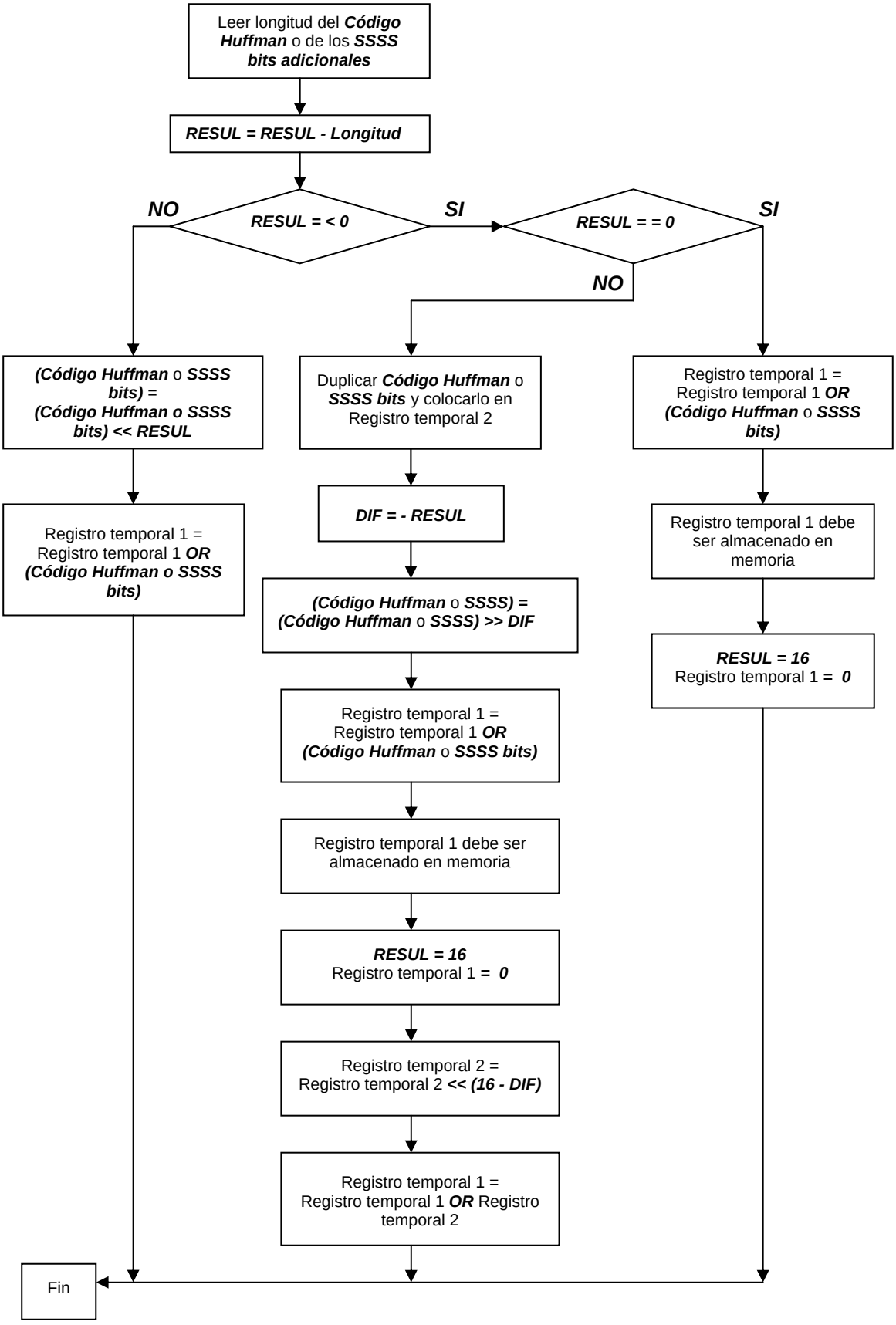


Figura 4.12.- Algoritmo Rutina de concatenación

Estas tablas de código junto con sus longitudes son encargadas de interactuar con la rutina de concatenación para efectuar la codificación entrópica. El criterio principal de operación de la rutina de concatenación, es utilizar dos registros temporales de 16 bits, así como conocer el número de bits disponibles que aún se pueden albergar en dichos registros.

Para conocer el número de bits que están disponibles en el registro temporal se utiliza la variable RESUL que es la encargada de dicho conteo el primer paso es inicializar la variable RESUL con 16 y limpiar el registro temporal 1, asignándole el valor de 0.

Después se debe realizar la lectura del código Huffman y su correspondiente longitud (para darle mayor flexibilidad a esta rutina, esta es capaz de leer los *SSSS* bits adicionales como datos de entrada) se realiza la operación RESUL – longitud, almacenando el resultado en la variable RESUL.

Con base en el valor que presente RESUL, *< 0, == 0,* ó *> 0* se realizarán tres diferentes procesos para cada opción.

Si *RESUL > 0*

- La variable que contiene al código Huffman o los SSSS bits adicionales, es desplazada a la izquierda por el número de bits que indique la variable RESUL, el resultado se almacena en la misma variable.
- Después se realiza la operación lógica OR entre el registro temporal 1 y la variable que fue desplazada, el resultado de esta operación se almacena en el registro temporal 1.
- El proceso finaliza, pero la variable RESUL y el registro temporal 1 conservan los valores que tienen al momento de salir de la rutina.

Si *RESUL = = 0*

- Se realiza la operación lógica OR entre el registro temporal 1 y la variable que contiene el código Huffman o los SSSS bits adicionales, el resultado se almacena en el registro temporal 1.
- Se almacena el registro temporal 1 en la memoria donde se deben colocar los datos comprimidos.
- RESUL = 16 y registro temporal 1 = 0, tanto el registro temporal 1 como la variable RESUL conservan sus valores al salir de la rutina.

Si *RESUL < 0*

- El código Huffman o los SSSS bits adicionales deben ser almacenados en el registro temporal 2.
- Se efectúa la operación DIF = - RESUL.
- La variable que contiene el código Huffman o los SSSS bits es desplazada a la derecha por el número que indica DIF el resultado es almacenado en la misma variable.
- Después se realiza la operación lógica OR entre el registro temporal 1 y la variable que fue desplazada, el resultado de está operación se almacena en el registro temporal 1.

- El registro temporal 1 se almacena en memoria donde se colocan los datos comprimidos.
- RESUL = 16 y registro temporal 1 = 0
- El registro temporal 2 es desplazado a la izquierda por el resultado de la operación (RESUL - DIF), el resultado es almacenado en el mismo registro temporal 2.
- Se realiza la operación lógica OR entre el registro temporal 1 y el registro temporal 2, el resultado es colocado en registro temporal 1.
- El proceso finaliza y el registro temporal 1 y RESUL deben conservar los valores con los cuales terminan.

Para ilustrar este procedimiento se supondrá que se deben codificar los siguientes elementos:

-22, 0, 0, 0, 5, en este caso el valor *DPCM* es -22, los términos restantes son de *AC*, todos pertenecientes a una señal de luminancia. El valor de -22 tiene un código Huffman de 110, *SSSS bits adicionales* = 01001, longitud de código = 3 y longitud *SSSS bits adicionales* = 5.

Entonces RESUL = 16, registro temporal 1 = 0, codificándose primero el código Huffman cuya longitud es 3, entonces se realiza la operación

RESUL = RESUL – 3 = 13

Se realiza la comparación RESUL > 0 *(13 > 0)*, con lo cual la ejecución del algoritmo fluye por la rama izquierda de la ***Figura 4.12***. El código Huffman *DC* está almacenado de la siguiente manera:

0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**0**

Debe ser desplazado 13 unidades a la izquierda, quedando:

1	**1**	**0**	0	0	0	0	0	0	0	0	0	0	0	0

Se realiza una OR lógica con el registro temporal 1(cuyo valor es cero).

1	**1**	**0**	0	0	0	0	0	0	0	0	0	0	0	0

OR

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

El resultado queda de nuevo en el mismo registro temporal 1

1	**1**	**0**	0	0	0	0	0	0	0	0	0	0	0	0

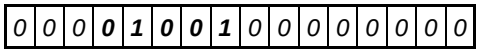
El siguiente paso es concatenar los *SSSS bits adicionales* 01001 para formar el código total de *DC*, la longitud de los *SSSS bits* es 5. Nuevamente

RESUL = 13 – 5 = 8

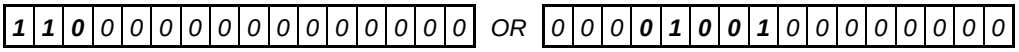
RESUL > 0 *(8>0)*, por tanto el algoritmo ejecuta nuevamente la misma rama, los *SSSS bits adicionales* están almacenados de la siguiente manera:

0	0	0	0	0	0	0	0	0	0	0	**0**	**1**	**0**	**0**	**1**

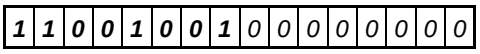
Los cuales deben ser desplazados ocho unidades a la izquierda.



Se realiza la OR lógica con el valor del registro temporal 1.



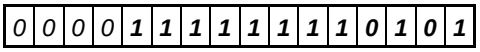
El resultado queda de nuevo en el registro temporal 1.



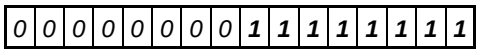
El siguiente paso es concatenar el primer coeficiente de *AC*, su código Huffman es 111111110101 y su longitud es de 12. Entonces

RESUL = 8 – 12 = -4

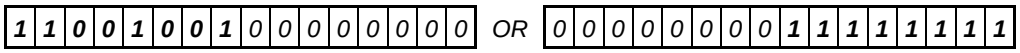
RESUL < 0 *(-4<0)*, por tanto el algoritmo ejecuta las operaciones de su rama central. El registro temporal 2 es almacenado con el código Huffman.



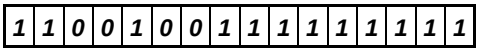
Estos datos son copiados y se desplazan a la derecha de acuerdo al resultado de DIF = - (-4) = 4.



Se realiza la OR lógica con el valor anterior y él del registro temporal 1.



El resultado de dicha operación es:

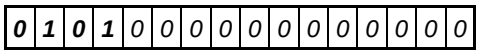


El registro temporal 1 se almacena en la memoria del *DSP*, esto se logra gracias a la declaración del arreglo *Datos_comprimidos [contador]*, (que es el arreglo encargado de almacenar la información comprimida) en este caso el registro temporal 1 se almacena en el primer elemento del arreglo *Datos_comprimidos*, cuando el registro temporal 1 vuelva a llenarse su almacenamiento se deberá realizar en el siguiente elemento del arreglo.

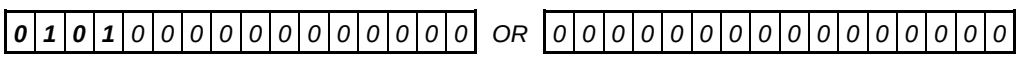
Después RESUL = 16, registro temporal 1 = 0, además el registro temporal 2 debe ser desplazado a la izquierda por la cantidad que indica el resultado de:

RESUL – DIF = 16 – 4 = 12

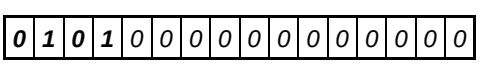
Quedando como



Se realiza la OR lógica entre el registro temporal 1 y el registro temporal 2.



El resultado de la OR lógica, es:



El siguiente paso es concatenar los *SSSS bits adicionales* del coeficiente AC, estos son 101, cuya longitud es 3, en este caso

RESUL = 12 – 3 = 9

Nuevamente RESUL > 0 *(9 > 0)*, entonces el algoritmo se vuelve a ejecutar por la rama de la izquierda

0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**0**	**1**

El cual debe ser desplazado 9 unidades a la izquierda:

0	0	0	0	**1**	**0**	**1**	0	0	0	0	0	0	0	0	0

Se realiza la OR lógica con el valor anterior y él del registro temporal 1.

0	**1**	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0

OR

0	0	0	0	**1**	**0**	**1**	0	0	0	0	0	0	0	0	0

El Resultado de la OR lógica, se almacena en él registro temporal 1.

0	**1**	**0**	**1**	**1**	**0**	**1**	0	0	0	0	0	0	0	0	0

Así es como se termina la codificación para los primeros dos códigos entrópicos.

4.2.8.- Rutina para encontrar categoría SSSS DPCM

Antes de pasar a describir el algoritmo de codificación de los valores *DPCM*, se presenta la rutina que nos permite encontrar la categoría DPCM, su algoritmo se muestra en la ***Figura 4.13*** [6] [13].

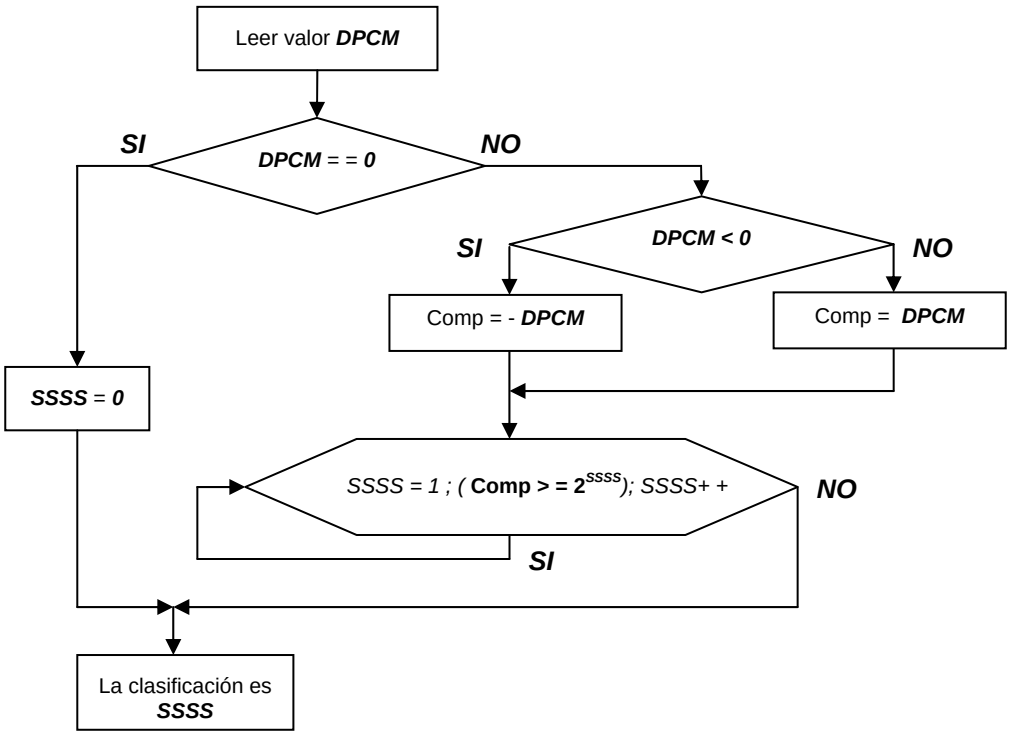


Figura 4.13.- Algoritmo para encontrar la clasificación de los valores DPCM

El primer paso es leer el valor *DPCM*, si este es igual a cero entonces se le asigna el valor de cero a la variable SSSS y se da por terminado el proceso. En caso contrario el valor absoluto de *DPCM* es asignado a la variable COMP.

El siguiente paso es asignar SSSS = 1, entrando a un ciclo de repetición que tiene como condición la comparación de la variable COMP con el resultado de 2^{SSSS} si la condición se cumple el valor de la variable SSSS se incrementa en 1, y el proceso se repite hasta que la condición no se cumpla, cuando la condición no se cumple la categoría que le corresponde a *DPCM* es el valor que contiene la variable SSSS.

4.2.9.- Rutina para encontrar categoría de los coeficientes AC

El algoritmo utilizado para encontrar la categoría *SSSS* de los coeficientes de *AC* se muestra en la ***Figura 4.14***, es muy parecido al algoritmo para encontrar la categoría *DPCM*, solo que en este último se elimina la opción que indica que los coeficientes de *AC* puedan ser iguales a cero [6] [13].

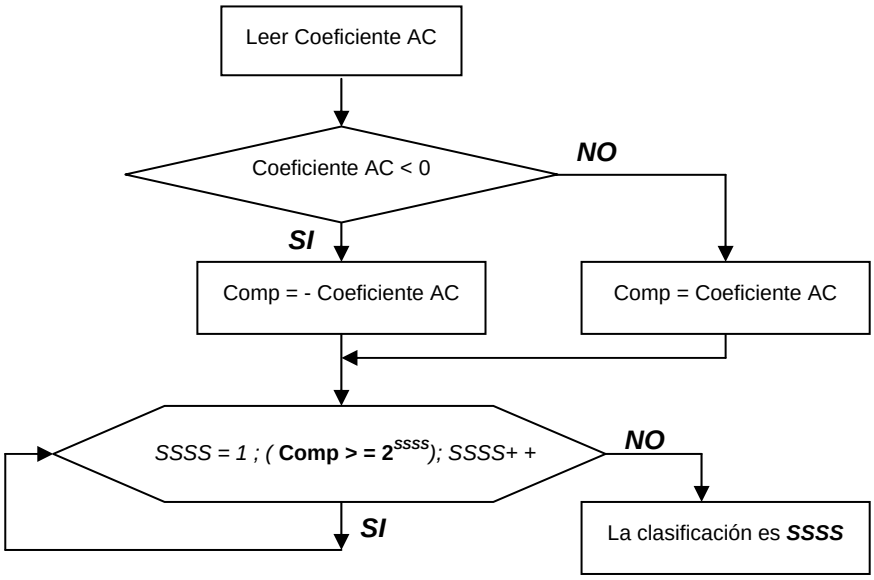


Figura 4.14.- Algoritmo para encontrar la clasificación de valores AC

4.2.10.- Rutina para encontrar SSSS bits adicionales

La última función que se necesita para realizar la codificación entrópica es la rutina para encontrar los *SSSS bits adicionales*, está se muestra en la ***Figura 4.15***, la creación de esta rutina es necesaria debido al hecho de que cuando se presenta un valor negativo, los bits restantes que se encuentran en el registro también son afectados por un signo negativo [6] [13].

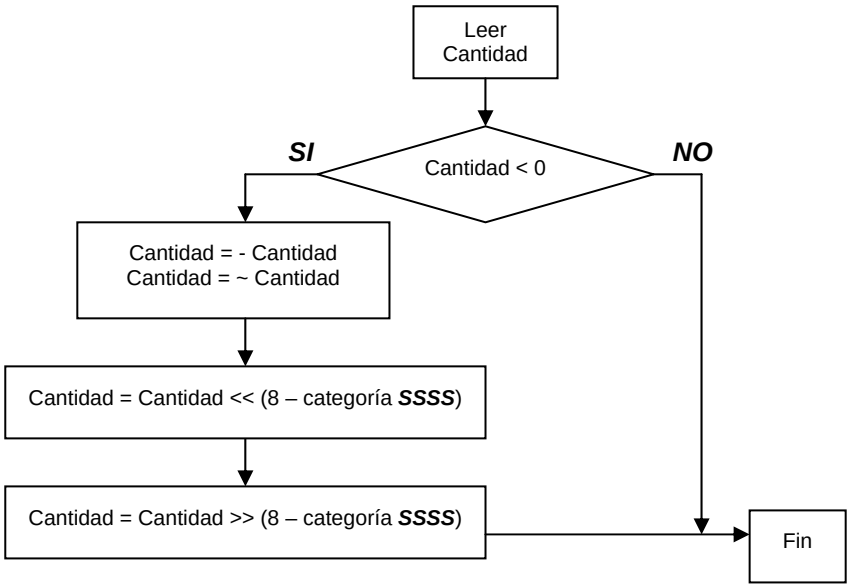


Figura 4.15.- Algoritmo para encontrar SSSS bits adicionales

Por ejemplo el valor de 5 en un registro de ocho bits es:

0	*0*	*0*	*0*	*0*	***1***	***0***	***1***

El valor de -5 está dado por el complemento a uno de 5 y en un registro de ocho bits se almacena como:

1	*1*	*1*	*1*	*1*	***0***	***1***	***0***

Para que funcione la rutina de concatenación se necesitan eliminar los bits de valor uno que se generan al utilizar el complemento a uno en un número negativo; por lo tanto es preciso desplazar a la izquierda el registro que contiene dicho valor en complemento a uno. El número de veces que se debe desplazar este registro a la izquierda está dado por # total de bits en el registro – *categoría SSSS*, en este caso como se encuentran en un registro de 8 bits y su *categoría SSSS* es tres, se tiene que 8 – 3 = 5 es el número de veces que debe desplazar a la izquierda.

0	***1***	***0***	*0*	*0*	*0*	*0*	*0*

La siguiente operación es correr hacia la derecha el registro anterior, nuevamente el número de veces que se debe desplazar está dado por # total de bits en el registro – *categoría SSSS*.

0	*0*	*0*	*0*	*0*	***0***	***1***	***0***

Así es como se obtienen lo *SSSS bits adicionales*, cuando se presenta un valor positivo no es necesario utilizar los corrimientos.

4.2.11.- Etapa de codificación DPCM

Como ya se ha mencionado los valores *DPCM* se codifican de manera distinta a los coeficientes de *AC*, el algoritmo utilizado para la realización de la codificación entrópica *DPCM* se muestra en la ***Figura 4.16***.

Como se ve en la ***Figura 4.16***, el algoritmo es totalmente secuencial, el primer paso es leer el valor *DPCM*, después se procede a encontrar la *categoría SSSS* por medio de su función correspondiente. Una vez que se encuentra dicho valor es posible asignar el código Huffman proveniente del arreglo de valores de *DC* así como su longitud, que proviene del arreglo de *Longitudes_de_DC*, a la rutina de concatenación.

Al finalizar está rutina se encuentran los *SSSS bits adicionales*, los bits adicionales así como la *categoría SSSS* son los valores que se deben pasar hacia la rutina de concatenación, al finalizar está rutina la codificación entrópica *DPCM* termina y solo resta la codificación de los coeficientes *AC* [13].

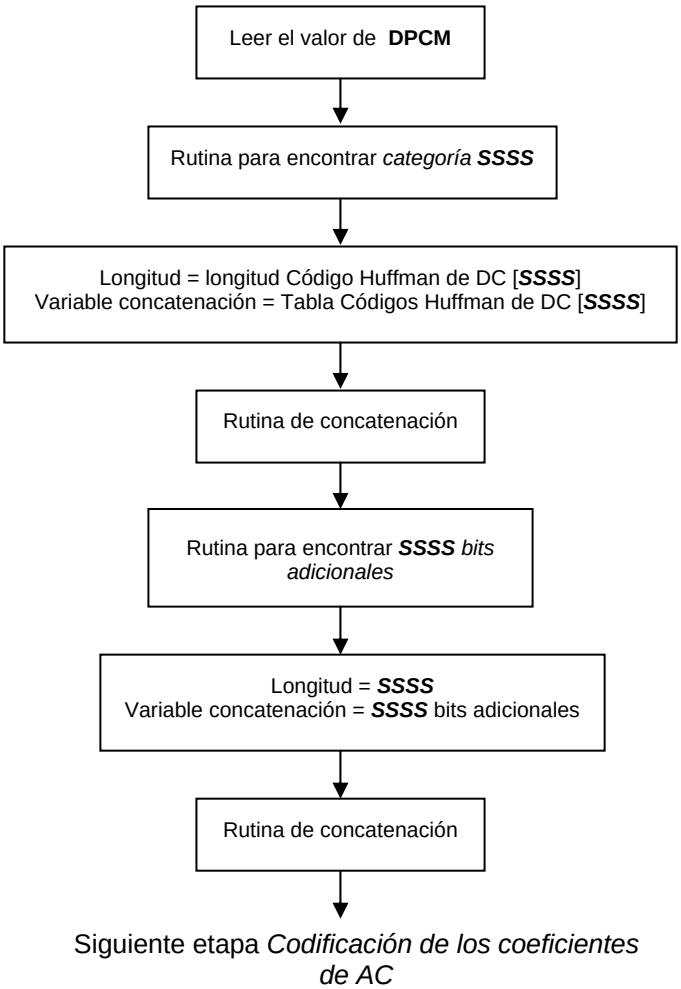


Figura 4.16.- Algoritmo de codificación de valores DPCM.

4.2.12.- Etapa de codificación AC

Como se observó en la ***Sección 3.3.4.6*** los coeficientes de *AC* se codifican de acuerdo al número de coeficientes de valor cero que anteceden a un coeficiente diferente de cero. Además existe un código especial *F/0* el cual indica la existencia de 16 coeficientes consecutivos de valor cero, debido a esto una *unidad de datos* puede contener tres códigos *F/0* como máximo. Así mismo al finalizar la codificación de una *unidad de datos* se le debe concatenar el código *EOB*. Para la realización de estos procesos se implementa el algoritmo de la ***Figura 4.17***.

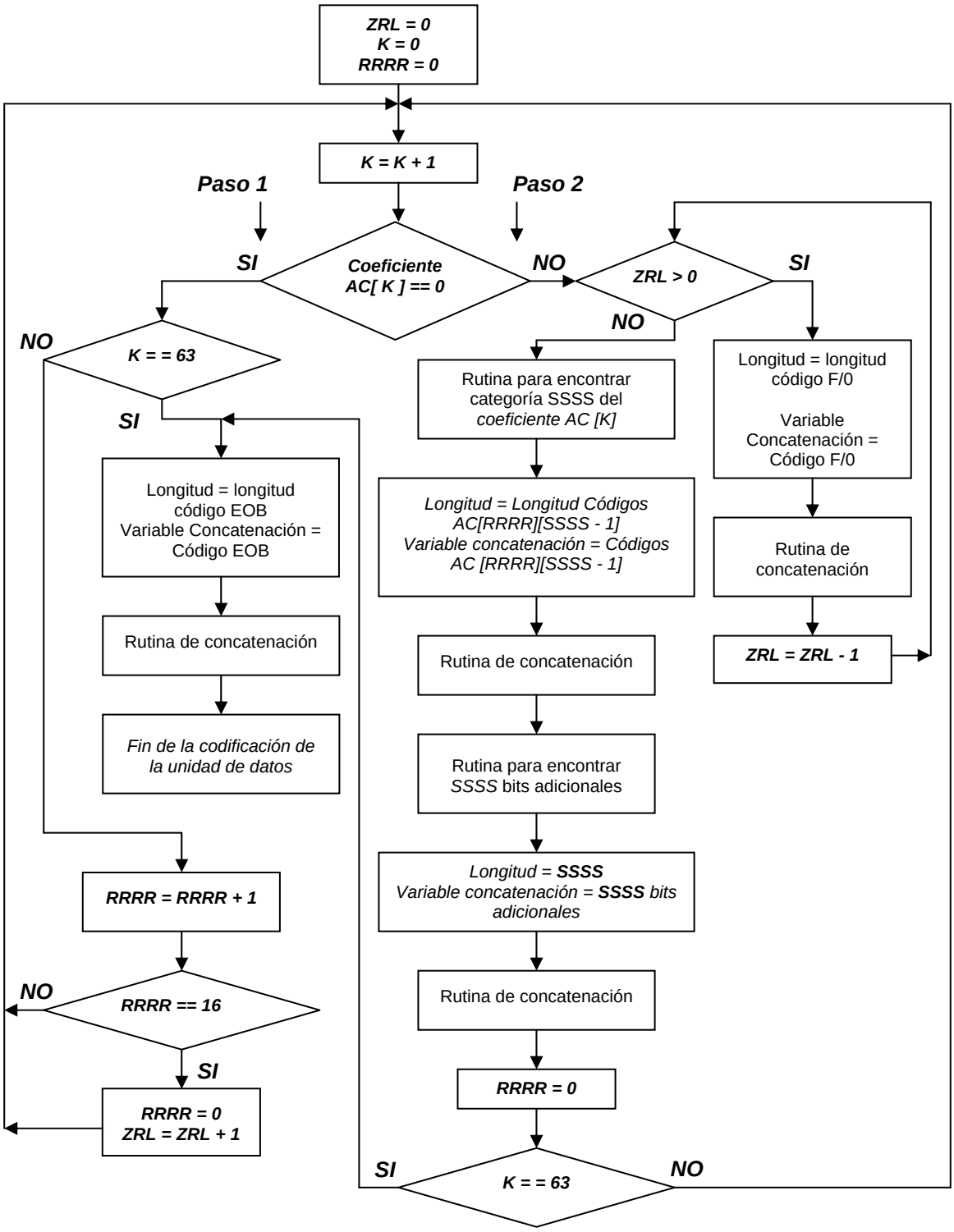


Figura 4.17.- Diagrama de flujo del algoritmo de codificación de coeficientes AC

Las variables principales de este algoritmo son ZRL, bandera encargada del conteo del número de veces que se repite el código *F/0*, su valor máximo puede ser tres. RRRR variable que contabiliza el número de coeficientes de valor cero consecutivos que preceden a un coeficiente diferente de cero. K es la variable encargada de realizar el barrido del arreglo unidimensional que previamente fue ordenado en Zigzag.

- Se inicializa ZRL, RRRR y K a cero.
- Se incrementa K en uno y se fórmula la sentencia *Coeficiente AC [K] = = 0.*

Paso uno: Corresponde al hecho de que los coeficientes *AC* tengan un valor de cero. Si la variable K tiene valor igual a 63, significa que es el último elemento del arreglo unidimensional, por tanto la codificación ha terminado y se debe de concatenar el código *EOB*. En caso de que el valor de K no sea 63, la bandera RRRR debe ser incrementada en 1, para formular la sentencia RRRR = = 16, entonces si la respuesta es afirmativa, RRRR toma el valor de cero y ZRL se incrementa en uno, lo cual indica que las condiciones del código *F/0* están presentes en el arreglo, después se regresa a paso dos. Si el valor de RRRR no es 16, entonces se debe regresar a paso dos.

Paso dos: Corresponde al hecho de que los coeficientes *AC* tienen un valor que no es cero. Se formula la sentencia ZRL > 0 entonces si la respuesta es afirmativa se debe concatenar el código *F/0* y disminuir el valor de ZRL en 1, este proceso continuará hasta que el valor de ZRL sea cero.

En caso de que el valor de ZRL, sea cero (ya sea porque ha disminuido en el ciclo o porque al momento de realizar la comparación no presentaba este valor) primero se debe encontrar la categoría SSSS del coeficiente que se está procesando, después, es necesario encontrar el código Huffman, que está dado por RRRR y SSSS; una vez que se encuentra el código se asigna a la rutina de concatenación, inmediatamente después, se encuentran los *SSSS bits adicionales* y se vuelve a aplicar la rutina de concatenación.

A continuación se realiza la sentencia K == 63, entonces si K presenta este valor se debe concatenar el código *EOB*, pues la codificación ha finalizado. En caso contrario se debe regresar al paso dos. Una vez que se ha alcanzado el valor del último coeficiente de *AC* no importa el valor que tenga RRRR o ZRL, la codificación debe terminar [13].

4.2.13.- Concatenación del marcador EOI

Una vez que todas las *unidades de datos* de una imagen de la secuencia de video se han codificado, el código *EOI 1111111110000001* (Fin de Cuadro) debe ser concatenado. Este marcador es utilizado para indicar en el flujo de datos comprimidos que un cuadro de la secuencia de video está totalmente codificado, este código tiene 16 bits y solo es utilizado para este propósito [13].

4.3.- Algoritmo para la implementación de la decompresión MJPEG

De la misma manera que se describió en la ***sección 4.2*** el algoritmo general para la compresión *MJPEG*, en esta sección se procede a describir el algoritmo que realiza la decompresión *MJPEG.*

El funcionamiento del decodificador *MJPEG*, se debe realizar tomando en cuenta que los datos de entrada, estarán contenidos en el arreglo *Datos_comprimidos.* Los elementos de dicho arreglo están formados por 16 bits, como se describió en la ***sección 4.2.7*** -la memoria *SDRAM* está formada por localidades de 32 bits, pero el ambiente *CCS* nos da la posibilidad de manejar registros de 8, 16 o 32 bits, por lo tanto en una localidad de la *SDRAM* es posible almacenar dos elementos del arreglo *Datos_comprimidos[contador]*, esto se ejemplifica en la ***Figura 4.18*** -.

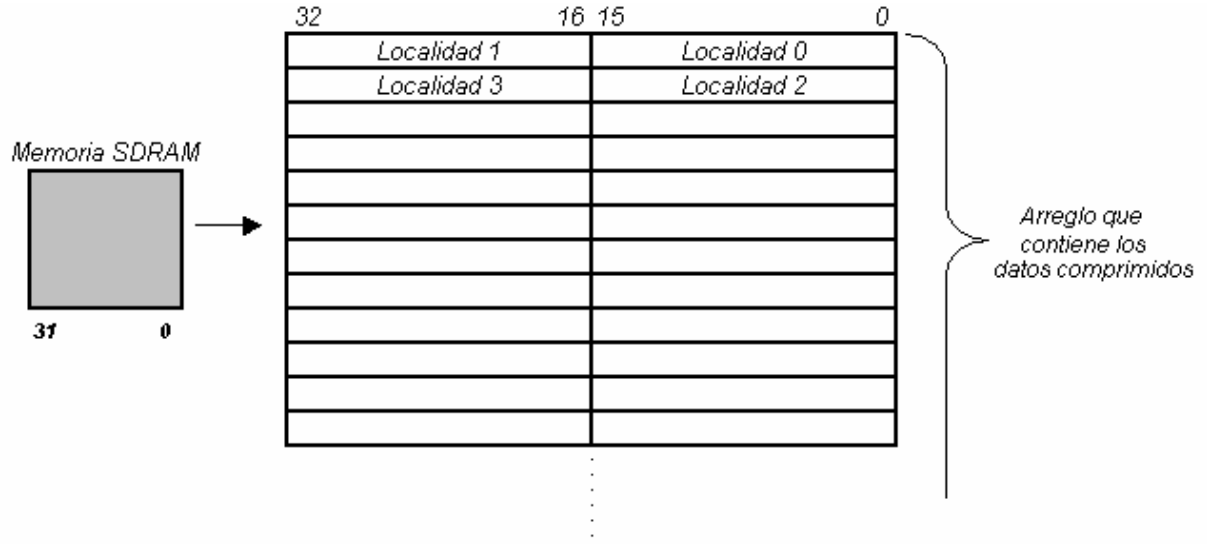


Figura 4.18.- Almacenamiento en memoria del arreglo que contiene los datos comprimidos

El algoritmo utilizado en para la realización de la decompresión *MJPEG* se muestra en la ***Figura 4.19***, el cual es muy similar al algoritmo correspondiente para la implementación del codificador *MJPEG*. El código total utilizado para realizar la implementación de la etapa de decompresión se cita en el ***Anexo E***.

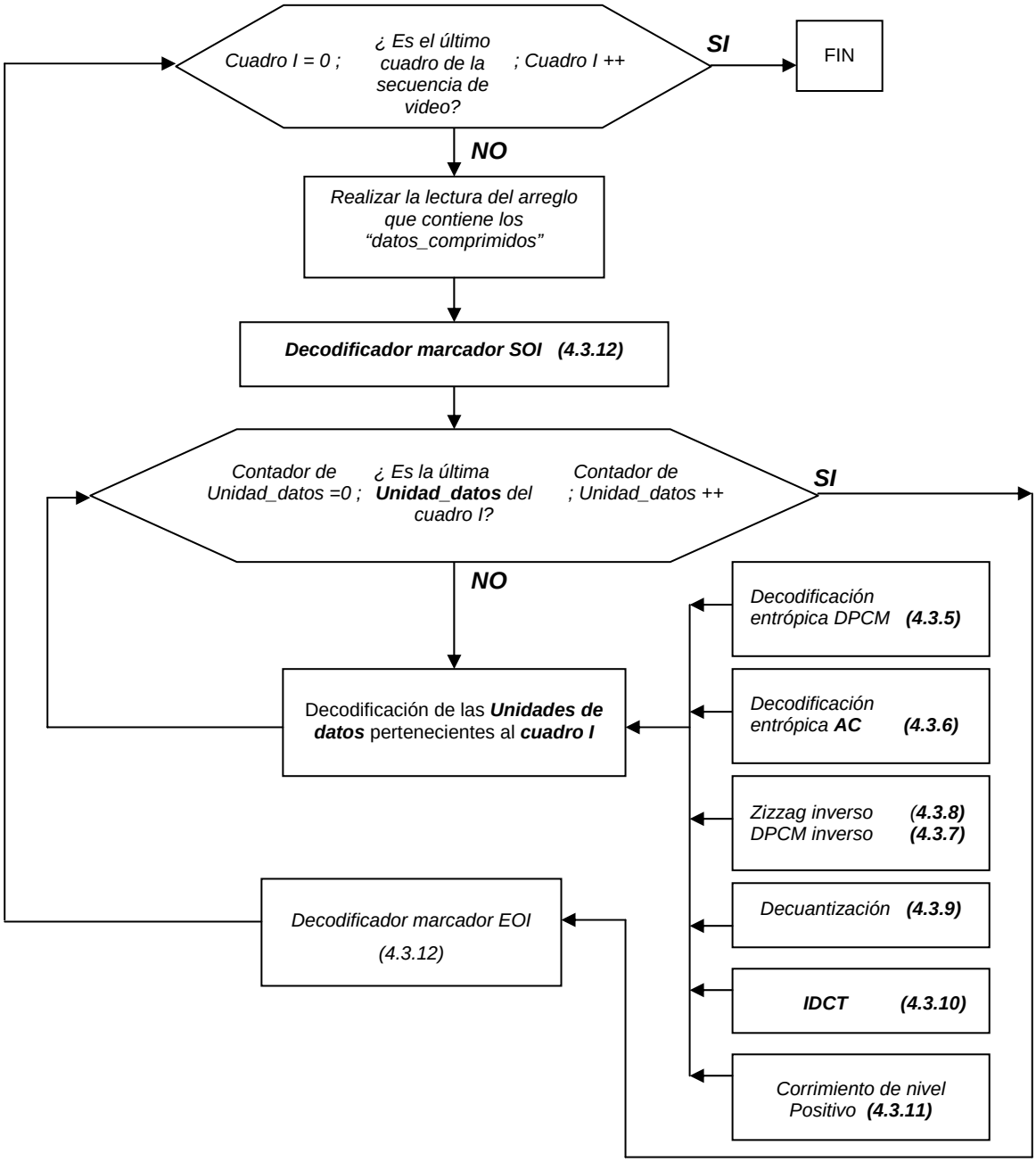


Figura 4.19.- Algoritmo general de Decompresión MJPEG utilizado en la implementación del sistema.

En la decompresión *MJPEG* la variable *Cuadro I* es la encargada de realizar el conteo de los cuadros que se deberán decodificar. El siguiente paso es decodificar el marcador *SOI* el cual nos indica el inicio del cuadro, posteriormente se procede a realizar la decodificación de una *unidad o unidades de datos* (Dependiendo del número de componentes), este proceso consta de los bloques de ejecución (IDCT, Decuantización, etc). Una vez que se ha decodificado una *unidad de datos* ésta debe ser asignada a un espacio de memoria adecuado donde se almacenarán las imágenes reconstruidas, posteriormente se procede a ejecutar el bloque condicional que pregunta si es la última *unidad de datos*, si la respuesta es negativa se procede a realizar la decodificación de la siguiente *unidad de datos*, si la respuesta es positiva se procede a decodificar el marcador *EOI*, el cual indica que ha finalizado la decodificación de un cuadro de la secuencia de video.

Por último se pregunta si es el último cuadro de la secuencia, si la respuesta es negativa se procede a incrementar en uno el valor de la variable *Cuadro I* y se procede a decodificar de nuevo el marcador *SOI*, si la respuesta es afirmativa el proceso de decodificación *MJPEG* finaliza. Una descripción mas detallada de los bloques que forman este algoritmo se darán en las siguientes secciones.

4.3.1.- Rutina de lectura de bits

De acuerdo al desarrollo de la ***sección 4.2.7*** los datos comprimidos son almacenados en localidades de memoria del *DSP*, estos datos pueden ser manipulados mediante la declaración de un arreglo que apunte a dichas localidades de memoria. El problema de dicho arreglo es que contiene tanto los códigos Huffman como los *SSSS bits adicionales* de manera concatenada, para poder acceder a dicha información se crea una *rutina de lectura de bits,* la cual nos permite leer bit a bit cada elemento del arreglo *Datos_comprimidos*. El algoritmo de lectura de bits se muestra en la ***Figura 4.20*** [6].

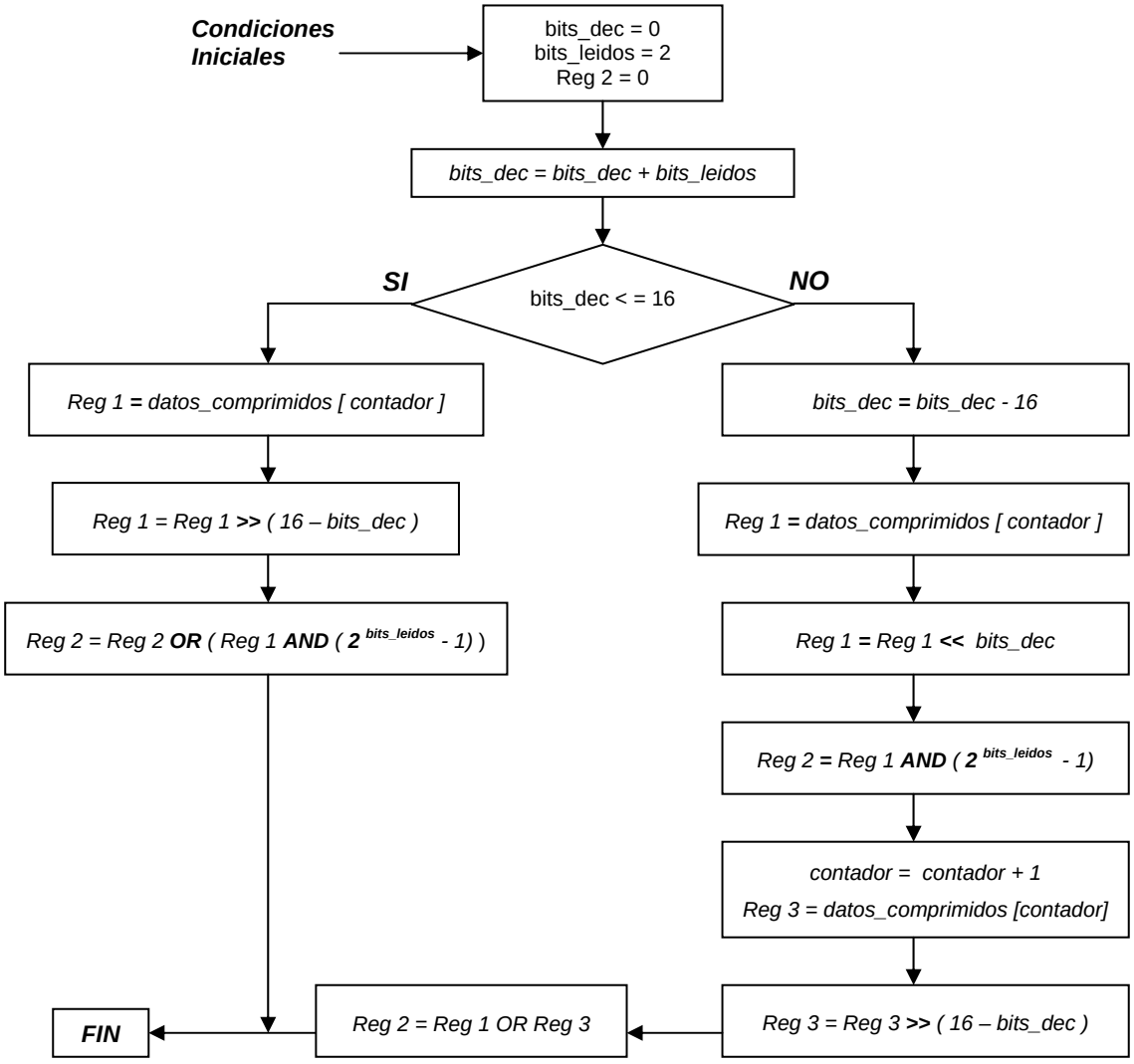


Figura 4.20.- Algoritmo de lectura de bits

El criterio principal para utilizar dicho algoritmo es el siguiente:

- Una cadena de bits que bien puede representar un código Huffman o los SSSS bits adicionales, puede encontrarse en una sola localidad de memoria o en dos localidades consecutivas.

Debido a esto el algoritmo se divide en dos partes, cuando la cadena de bits que nos interesa se encuentra en una solo localidad de memoria y cuando se encuentra en dos.

Caso 1: La cadena de bits leída se encuentra en una localidad de memoria de 16 bits.

1	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X

Si estos bits pertenecen al primer elemento del arreglo *Datos_comprimidos,* la cadena de bits que esperamos decodificar le corresponde al código Huffman del valor *DPCM*. Del algoritmo de la ***Figura 4.20*** las variables que nos interesan son *bits_dec* que es la encargada de almacenar el número de bits que ya han sido decodificados en dicho elemento.

La variable *bits_leídos* es la encargada de indicar el número de bits que se habrán de leer; debido a que suponemos que es el primer elemento del arreglo *Datos_comprimidos*, las condiciones iniciales son *bits_dec = 0, bits_leídos = 2*, y *registro 2 = 0.*

El siguiente paso es realizar la operación:

$$bits\_dec = bits\_dec + bits\_leidos = 0 + 2 = 2$$

Este valor es menor que 16 por tal motivo se opta por la rama izquierda del algoritmo, después se tiene que el valor del *registro 1* es:

1	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X

A continuación se procede a realizar un corrimiento hacia la derecha del registro 1, este corrimiento está dado por el resultado de *16 – bits_dec* = 16 – 2 = 14, con lo cual el registro 1 queda:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Posteriormente se procede a realizar una operación de enmascaramiento de bits entre el registro 1 y la máscara que está dada por $2^{bits\_leidos} - 1$, este proceso es necesario pues se desea eliminar los bits que anteceden a los bits que se leyeron y fueron corridos hacia la derecha.

El resultado de $2^2 - 1$ es 11, como la operación AND se realiza con un registro de 16 bits automáticamente se completan los bits restantes de la máscara. Por lo tanto

0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

AND

0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Da como resultado

0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Este resultado pasa a ser el operando de una operación lógica OR que se realiza con el registro 2

0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

OR

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

El resultado se almacena nuevamente en el registro 2 = *0000000000000011* y el proceso finaliza. Se puede observar que este valor es un código Huffman no válido por tal motivo es preciso volver a leer otro bit; pero antes de volver a ejecutar el algoritmo de *lectura de bits* es necesario ejecutar un bloque que nos permita modificar el valor de las variables *bits_dec* y *bits_leidos*, dicho bloque se muestran en la ***Figura 4.21***.

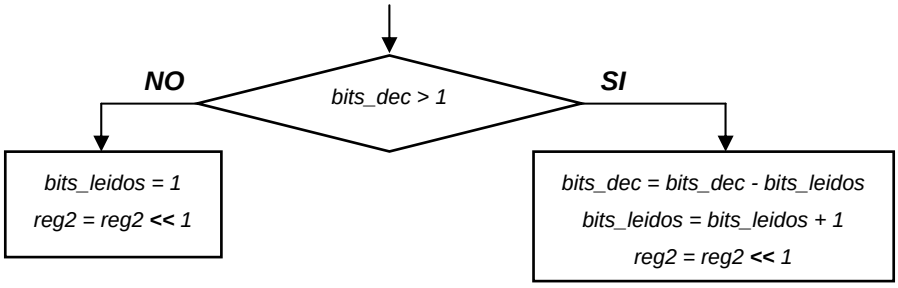


Figura 4.21.- Bloque para modificar las variables *bits_dec* y *bits_leidos*

Como *bits_dec > 1, (2 > 1)* entonces
bits_dec = 2 – 2 = 0;
bits_leidos = 2 + 1 = 3
registro 2 = registro 2 << 1 =

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Y ahora es posible volver a ejecutar el algoritmo de *lectura de bits,* se realiza de nuevo la suma *bits_dec* = 0 + 3 = 3, la comparación 3 < 16 indica que debe tomarse nuevamente la rama izquierda del algoritmo por lo tanto el registro 1, queda.

1	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X

Realizando las operaciones de corrimiento y enmascaramiento, el registro 1 toma el siguiente valor

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Se realiza la operación OR entre el registro 1 y el registro 2.

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

OR

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

El nuevo valor de registro 2 es *0000000000000111,* también es un código Huffman no válido por tal motivo es preciso volver a ejecutar otra vez el algoritmo de *lectura de bits.*

Como *bits_dec > 1, (3 > 1)* entonces
bits_dec = 3 – 3 = 0;
bits_leidos = 3 + 1 = 4
registro 2 = registro 2 << 1 =

0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0

Ejecutando nuevamente el algoritmo de lectura bits_dec = 0 + 4 = 4, se tiene que el valor final de registro 1 es *0000000000001110*, por último se realiza de nuevo la operación OR entre los dos registros.

0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0

OR

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0

Lo que da como resultado *0000000000001110* y este valor corresponde a un código Huffman válido de DC, por tal motivo no es necesario realizar otra lectura de bits.

Caso 2: La cadena de bits se encuentra en dos localidades de memoria consecutivas, tal como se muestra a continuación:

X	X	X	X	X	X	X	X	X	X	X	X	X	1	1

1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Se observa que el valor de *bits_dec* no es cero sino 14, por lo tanto se procede a asignar el valor de dos a la variable *bits_leidos.*

bits_dec = 14 + 2 = 16

La comparación es *(16 <= 16)*, por lo tanto se debe seguir la rama izquierda del algoritmo, el valor que es asignado al registro 1 es :

X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1

Realizando las operaciones que se enlistan en esta rama el valor final del registro 2 es:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Este valor no es un código Huffman válido, por tal motivo se ejecutan los bloques de incremento. *bits_dec > 1, (16 > 1)* entonces

bits_dec = 16 – 2 = 14 ;

bits_leidos = 2 + 1 = 3

registro 2 = registro 2 << 1 =

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Aplicando nuevamente el algoritmo de lectura se obtiene *bits_dec = 14 + 3 = 17* y se realiza la comparación *(17 <= 16)*, por tal motivo la ejecución del algoritmo de lectura de bits fluye ahora por su rama derecha y se tiene que *bits_dec* = 17 – 16 = 1.

El Registro 1 tiene el valor de *XXXXXXXXXXXXXX11,* este registro sufre un desplazamiento hacia la izquierda que está dado por *bits_dec=1*, nuevamente se aplica la máscara de bits sobre este registro y se obtiene:

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

A continuación se incrementa el valor del contador en uno; con lo cual se pasa a procesar el siguiente elemento del arreglo *Datos_comprimidos.* Se utiliza ahora el registro 3 al cual se le asigna el valor del elemento *Datos_comprimidos.*

1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Este registro debe ser desplazado hacia la derecha por el resultado dado por *16 -bits_dec,* lo cual da 16 – 1 = 15, quedando.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Finalmente se realiza la operación OR entre el registro 1 y el registro 3 el resultado es almacenado en registro 2.

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	*OR*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

El valor de registro 2 es por lo tanto *0000000000000111,* nuevamente se observa que está cadena de bits forma un código Huffman no válido, por lo tanto.

bits_dec > 1, (1 == 1) entonces

bits_leidos = 1

registro 2 = registro 2 << 1 =

0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0

Ejecutando otra vez el algoritmo de lectura se tiene que *bits_dec* = 1 + 1 = 2 ; realizando la comparación *(2 <= 16),* la ejecución del algoritmo vuelve a fluir por la rama izquierda, el valor que se obtiene de registro 1 es:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Por último se realiza la operación OR entre los registros 1 y 2

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	*OR*	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0

Con lo cual se encuentra nuevamente que el código Huffman es *0000000000001110*

4.3.2.- Rutina para encontrar código Huffman DC

Además de leer las cadenas de bits, también es preciso contar con método que nos permita saber si una cadena de bits leída es válida o no. Para determinar si una cadena es válida se debe tener en cuenta que un código Huffman tanto de *DC* como de *AC*, no puede contar solo con bits de valor uno, y ningún código puede estar formado por un solo bit. El procedimiento para encontrar un código válido es el siguiente, se deben conocer previamente la tabla de valores *DC* para los códigos Huffman la cual se muestra en la ***Tabla A.3*** (anexos) [6].

La ***Tabla A.3*** se puede escribir tal como aparece en la ***Tabla 4.5***, para decompresión de Luminancia.

SSSS_mínimo	*Longitud del código*	*Código mínimo*	*Código máximo*
0	*2*	*00*	*00*
1	*3*	*010*	*110*
6	*4*	*1110*	*1110*
7	*5*	*11110*	*11110*
8	*6*	*111110*	*111110*
9	*7*	*1111110*	*1111110*
10	*8*	*11111110*	*11111110*
11	*9*	*111111110*	*111111110*

Tabla 4.5.- Tabla de códigos Huffman DC reorganizada utilizada para la decompresión de Luminancia.

En la ***Tabla 4.5*** se observa que el único código que cuenta con dos bits es 00, por lo tanto su valor mayor es 00 a la vez que su valor menor y el valor de *SSSS_mínimo* es cero. Los códigos que están formados por tres bits son 010, 011, 100, 101, 110, por lo tanto para este grupo su valor menor es 010, su valor mayor es 110 y el valor *SSSS_mínimo* es 1.

El procedimiento para determinar si un código es válido se basa en los siguientes criterios:

- Determinar si la cadena de bits leída no consta solamente de bits de valor uno.
- El valor del número de bits leídos debe ser menor o igual al valor máximo del grupo cuya longitud es la misma que el número de bits leídos.
- Si los dos primeros párrafos se cumplen, se debe restar el valor del código menor del grupo de la cadena de bits leída.
- Para determinar el valor de *SSSS* es preciso sumar el resultado de la operación al valor *SSSS* del primer código que conste del número de bits de la cadena encontrada.

Por ejemplo supongamos que se tiene el código 100, por lo tanto:

Cadena de bits leidos (100) < = Valor máximo del grupo (110)

Condición que es verdad entonces

Cadena de bits leída - valor menor del grupo (*SSSS_mínimo*) = 100 – 010 = $2_{decimal}$

Por lo tanto el valor de *SSSS* que corresponde a dicho código es

SSSS = *SSSS_mínimo* + resultado anterior = 1 + 2 = 3

Este resultado se puede verificar en la ***Tabla A.3***, dicho valor nos indican el número de *SSSS bits adicionales* que forman la cantidad *DPCM*. El algoritmo general para encontrar el código Huffman *DC* se muestra en la ***Figura 4.22***.

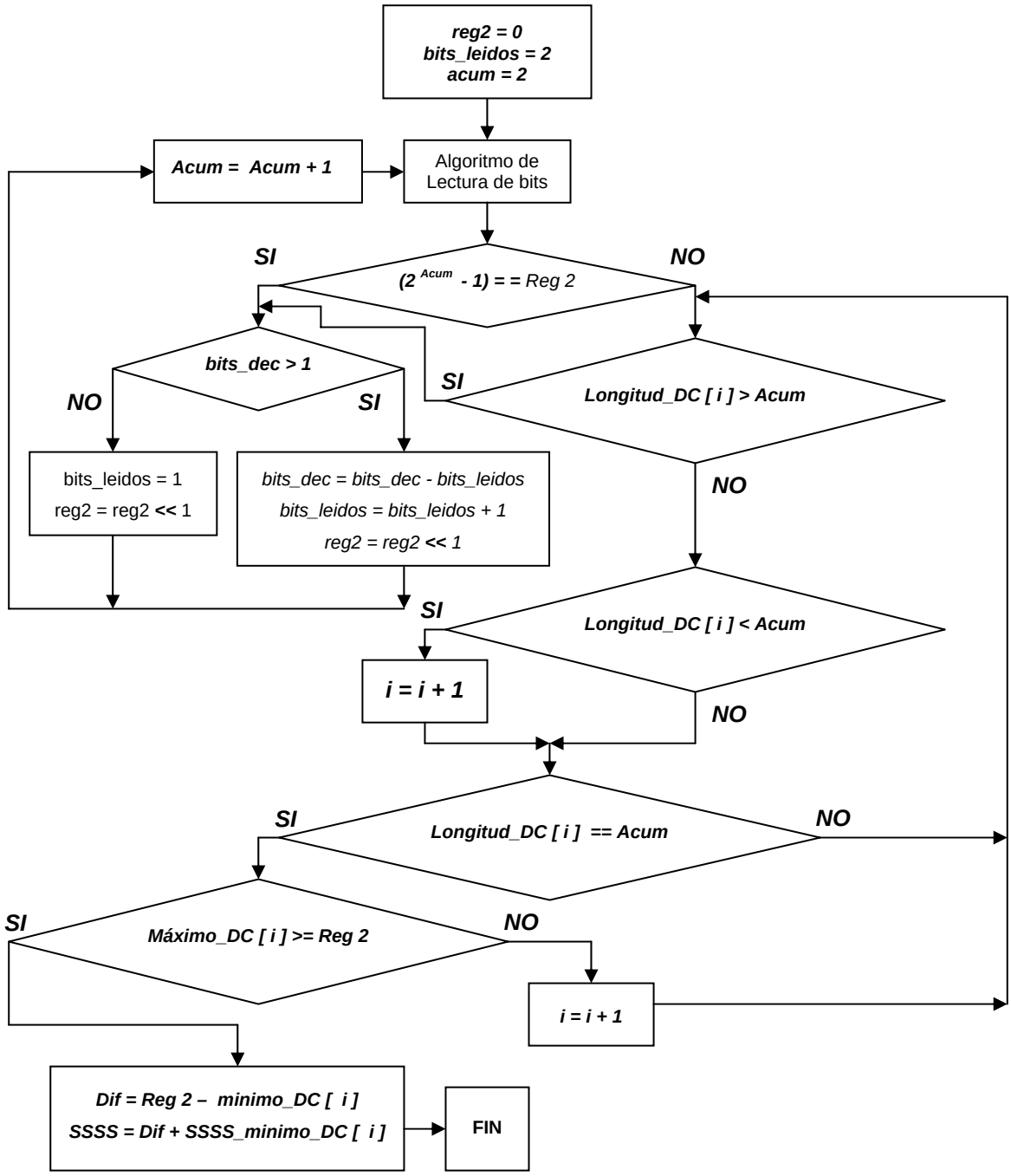


Figura 4.22.- Algoritmo para encontrar código Huffman DC

El primer paso en este algoritmo, es inicializar la variable ACUM = 2 e i = 0, la variable i es la encargada de realizar la búsqueda por los diferentes elementos de cada arreglo. El primer bloque condicional nos permite determinar si una cadena de bits leída consta solo de bits de valor uno, en caso de que la sentencia se cumpla es preciso volver a leer otro bit más en la cadena, por lo tanto es preciso aplicar el bloque para incrementar el valor de la variable *bits_leidos*, a su vez que se incrementa el valor de la variable ACUM.

En caso de que la sentencia no se cumpla significa que la cadena de bits leída consta de 0´s y 1´s por lo tanto se procede a corroborar su validez, para esto se utilizan una serie de bloques condicionales. Los tres primeros de ellos son una comparación entre el arreglo que contiene las longitudes de los códigos y la variable ACUM (la cual es la encargada de almacenar el número de bits con que cuenta una cadena) ; el último bloque condicional nos permite saber si el valor de la cadena de bits contenida en el registro 2 es menor o igual al valor máximo del grupo.

Una vez que los bloques condicionales finalizan y se encuentra que la cadena de bits leída no coincide con un código Huffman válido, es preciso volver a leer otro bit en la cadena, con lo cual se regresa al inicio del algoritmo; este proceso continuará hasta que la cadena de bits leída corresponda con un código Huffman válido.

En este ejemplo se ha considerado una señal de Luminancia, para encontrar el código Huffman de una señal de Crominancia es preciso utilizar sus respectivas tablas; los arreglos implementados en el *DSP*, para encontrar los códigos Huffman de *DC* se citan en las ***Tablas C.1*** a ***C.4*** y en las ***C.9*** a ***C.12***, de los anexos.

4.3.3.- Rutina para encontrar código Huffman AC

El criterio utilizado para encontrar los códigos de Huffman *DC* también debe ser aplicado para los códigos *AC*, solo que el procedimiento es distinto pues se debe tener en cuenta que los códigos de *AC* pertenecen a un arreglo bidimensional [6].

Un código de *AC* no solo indica el valor *SSSS* sino también nos señala el valor RRRR o número de coeficientes de valor cero que anteceden a un coeficiente distinto de cero, por lo tanto las matrices que contienen los códigos Huffman *AC* y sus respectivas longitudes nos permiten generar cuatro arreglos bidimensionales, los cuales contienen sus valores máximo, mínimo, longitud y *SSSS_mínimo*.

Por ejemplo se toman los códigos Huffman que corresponden a los primeros renglones de la ***Tabla A.5*** (anexos), es decir el caso que el valor RRRR es cero, se tiene que estos valores son: *00, 01, 100, 1011, 11010, 1111000, 11111000, 1111110110, 1111111110000010, 1111111110000011*.

Agrupándolos de acuerdo con sus valores máximos se tiene:

01, 100, 1011, 11010, 1111000, 11111000, 1111110110, 1111111110000011

Agrupándolos de acuerdo con sus valores mínimos se tiene:

00, 100, 1011, 11010, 1111000, 11111000, 1111110110, 1111111110000010

De acuerdo a su valor *SSSS_mínimo*

1, 3, 4, 5, 6, 7, 8, 9

Por último de acuerdo a su longitud se tiene

2, 3, 4, 5, 6, 7, 8, 10, 16

Este proceso continúa hasta el último renglón de la ***Tabla A.5*** (valor RRRR = 16), los arreglos resultantes completos implementados en el *DSP*, se citan en las ***Tablas C.5*** a ***C.8*** de los anexos.

Estos arreglos poseen dos columnas menos que los arreglos declarados para la compresión, pero hay que señalar que no todos los elementos de estos arreglos deben ser completados, por ejemplo el caso en que RRRR = 15, su código mayor será *1111111111111110* y la declaración en el arreglo es *Máximo [16][8] = { 1111111111111110, 0 , 0, 0, 0, 0, 0, 0 }*, en el caso del arreglo que contiene los valores de las longitudes el valor de un elemento no válido se declara con 17, que es una longitud que no posee ningún código Huffman.

Cuando se utiliza una señal de Crominancia, los arreglos resultantes tienen una columna extra, por tal motivo tienen 16 elementos más, estos arreglos se muestran en las ***Tablas C.13*** a ***C.16***.

El algoritmo general que nos permite encontrar el código Huffman *AC* se muestra en la ***Figura 4.23***. Dicho algoritmo es muy similar al utilizado para encontrar los códigos Huffman de *DC*, solo que ahora la búsqueda se debe realizar en un arreglo bidimensional por lo tanto es necesario utilizar dos índices i, j.

El primer bloque condicional nos permite saber si una cadena de bits leída consta solamente de bits de valor uno, los siguientes dos bloques de sentencias condicionales nos permiten conocer si el valor que se ha presentado corresponde a los códigos *EOB* o *ZRL*, en caso de que algunos de ellos se presente habrá de poner a uno sus respectivas banderas y el proceso finaliza.

En caso de que los códigos *ZRL* o *EOB* no se presenten se entra en una serie de bloques condicionales consecutivos, los cuales nos permiten saber si el valor de los bits leídos es igual a la longitud de algún código Huffman; en caso de que la longitudes coincidan se procede a comprobar si dicha cadena de bits es válida, en caso de que esto no suceda se regresa al principio de los bloques en cascada. En caso de que la cadena sea válida se realizan las operaciones necesarias para encontrar SSSS y RRRR, el valor de RRRR está dado simplemente por el valor i que indica el índice del renglón del arreglo.

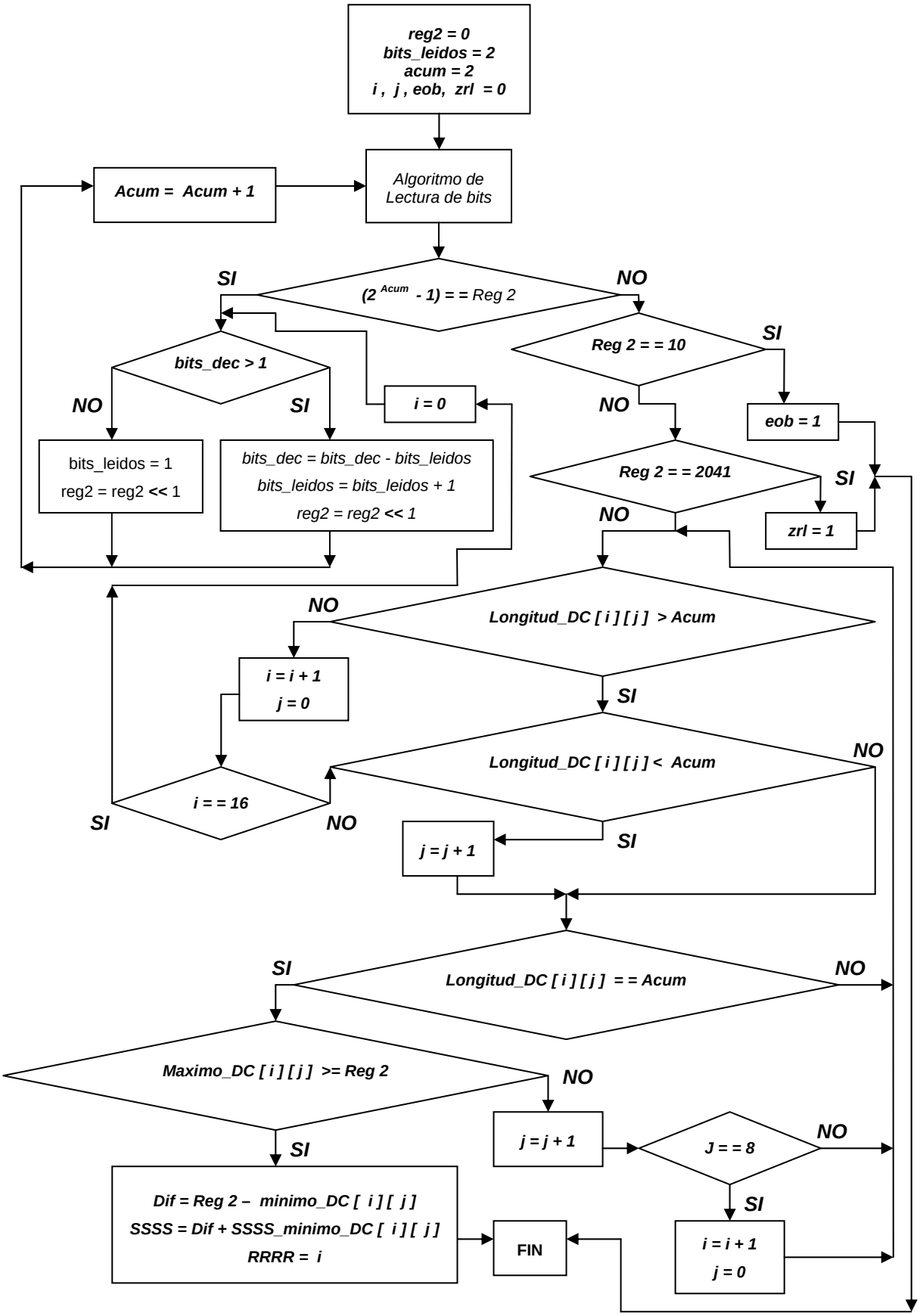


Figura 4.23.- Algoritmo para encontrar código Huffman AC

4.3.4.- Rutina para recuperar SSSS bits adicionales

El algoritmo que utiliza la subrutina para recuperar *SSSS bits adicionales* se muestra en la ***Figura 4.24***. En este caso al momento de inicializar la variable *bits_leidos* debe tomar el valor *SSSS*, para proceder a utilizar el algoritmo de lectura de bits, posteriormente el valor que nos entregue el registro 2 será asignado a las variables *comparación* y *cantidad.*

Para determinar si el valor contenido en el registro 2 es positivo o negativo se procede a realizar un corrimiento a la izquierda de la variable comparación dado por *(SSSS – 1),* con lo cual si el valor resultante es cero nos indica que es un valor negativo; si el valor resultante es uno implica que se la cantidad es positiva.

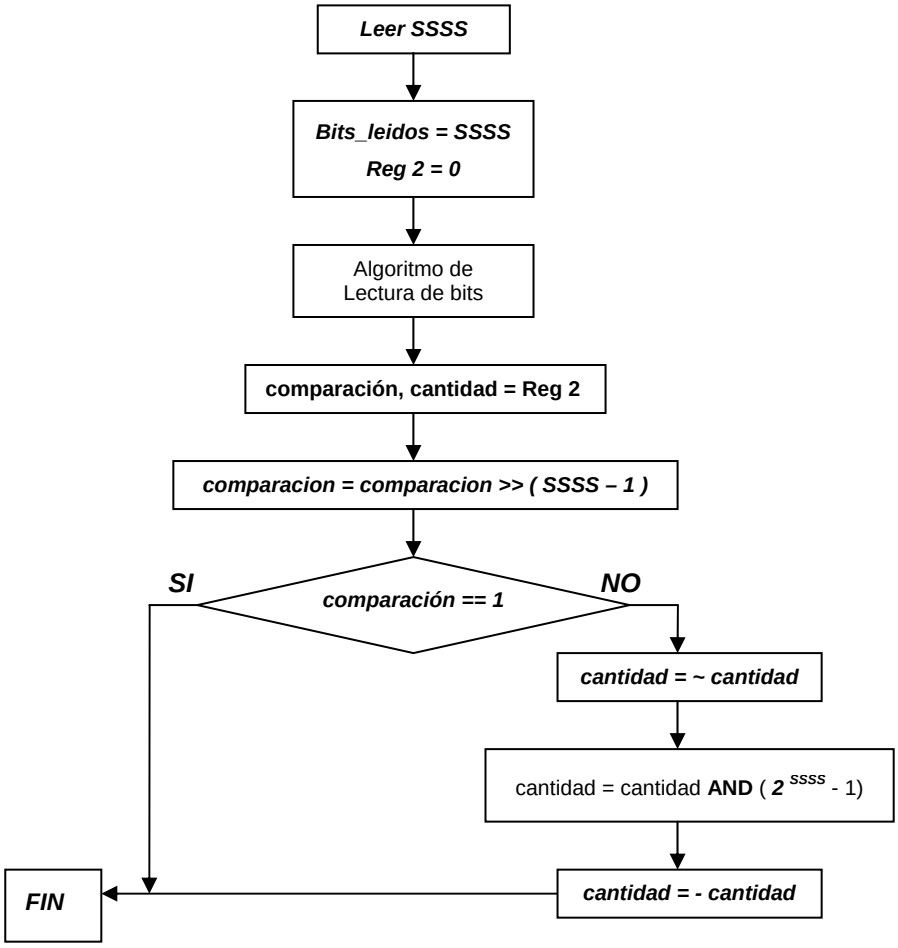


Figura 4.24.- Algoritmo para recuperar SSSS bits adicionales

Para obtener el valor de una cantidad negativa primero se realiza el complemento a uno de la variable cantidad, posteriormente se realiza la operación AND con la máscara 2^{SSSS}- 1, finalmente se multiplica por -1 el valor de la variable cantidad.

4.3.5.- Etapa de decodificación DPCM

El algoritmo que nos permite recuperar el valor *DPCM* de los datos comprimidos se muestra en la ***Figura 4.25***.

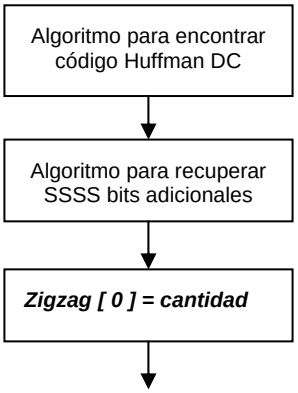


Figura 4.25.- Algoritmo para recuperar DPCM

El primer paso es ejecutar la rutina para encontrar el código Huffman *DC*, una vez que finaliza este proceso se procede a ejecutar el algoritmo para recuperar *SSSS bits adicionales*, finalmente la variable cantidad es asignada al primer elemento del arreglo *zig[64]* (que es el arreglo encargado de almacenar los elementos de la *unidad de datos* que están agrupados por el ordenamiento zigzag*)* [13].

4.3.6.- Etapa de decodificación AC

Para finalizar la decodificación entrópica de una *unidad de datos* se procede a la decodificación de los coeficientes de *AC*, la rutina que realiza dicho proceso se muestra en la ***Figura 4.26***.

El primer bloque consiste en encontrar el código Huffman *AC* posteriormente se verifica si la bandera *EOB* es igual a 1, en caso de que está condición se cumpla significa que el número de coeficientes restantes deben tener valor cero, para realizar esto se asigna a la variable RRRR el resultado de la operación 63 – k, siendo k el índice que controla el número de coeficientes restantes del arreglo *zig*, posteriormente se entra en un ciclo que incrementa el valor de k en uno, asigna el valor de cero al elemento *zig [k]* y se decrementa la variable RRRR en uno, este proceso continua hasta que la variable RRRR sea igual a cero y se procederá a poner a cero la bandera EOB y la variable k.

En caso de que el valor de la bandera *EOB* no sea uno se ejecuta un bloque condicional que pregunta si la bandera ZRL es uno, en el caso verdadero, la variable RRRR se le asigna el valor de 16 y se entra al ciclo de repetición que tiene como objetivo colocar 16 coeficientes de valor cero en el arreglo *zig*, al terminar este proceso la bandera ZRL es puesta a cero y se regresa al algoritmo para encontrar el código Huffman *AC*.

En caso de que la bandera ZRL no sea uno, se procede a ejecutar el algoritmo para recuperar los *SSSS bits adicionales* y se verifica si RRRR es igual a cero, en caso contrario se procede a colocar el número de coeficientes de valor cero indicados por RRRR, si RRRR es igual a cero se incrementará el índice de la variable en uno y se asigna la variable cantidad al elemento direccionado por la variable k; por último se retorna a la ejecución de la rutina para encontrar el código Huffman *AC*, de esta manera es como se logra decodificar totalmente una *unidad de datos* [13].

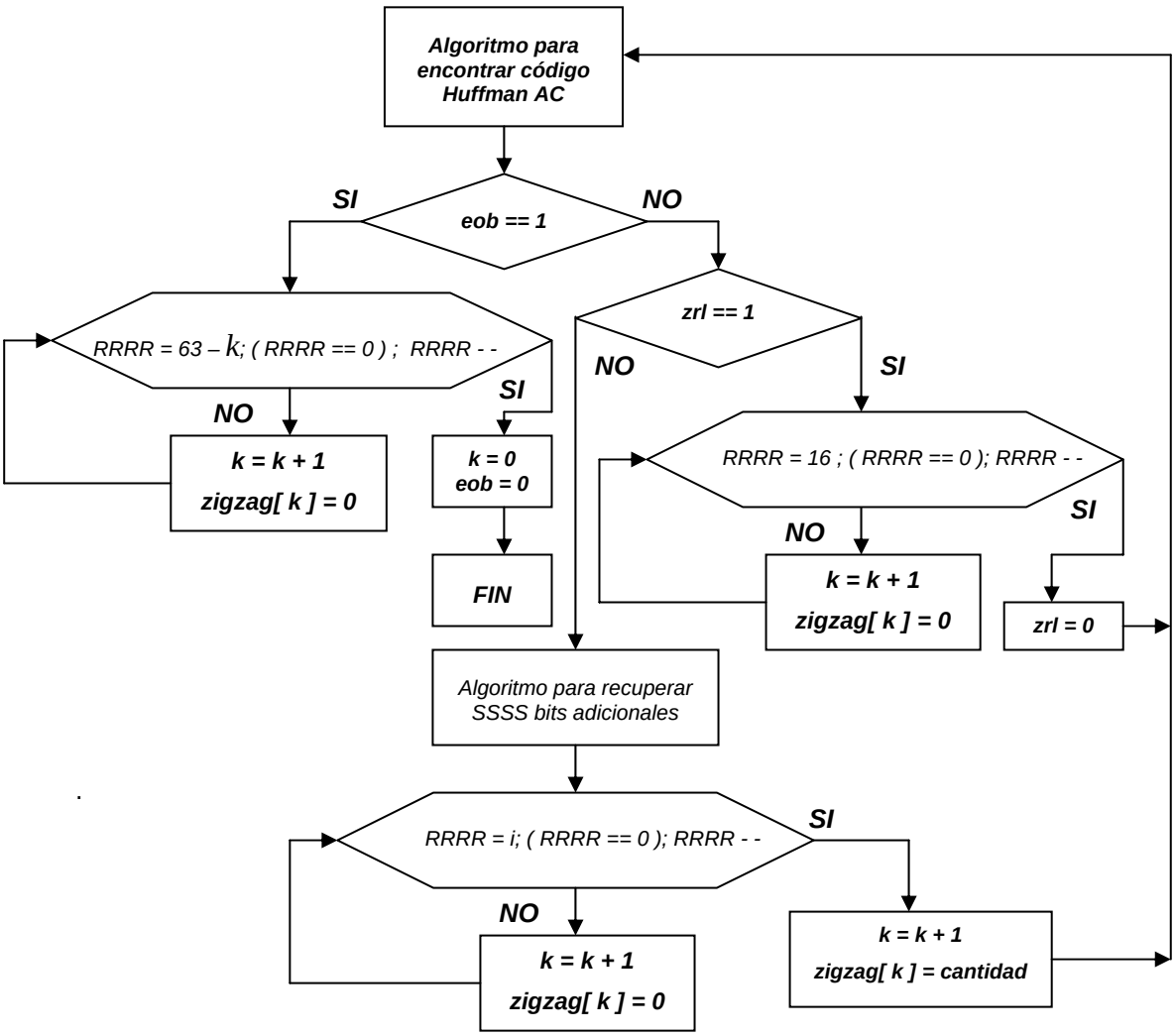


Figura 4.26 .- Algoritmo para recuperar coeficientes AC

4.3.7.- Obtención de los coeficientes de DC

Como se vio en la ***sección 3.3.4.3***, el coeficiente *DC* no es el valor codificado sino su diferencía *DPCM*, por tal motivo una vez que se ha decodificado una *unidad de datos,* el primer elemento de este arreglo debe sufrir un procedimiento inverso para poder encontrar el valor del coeficiente de *DC*, esto se realiza por medio de las siguientes líneas de código:

```
zig [0] = zig [0] + yn;
yn =zig[0];
```

Es preciso señalar que la primera vez que se ejecutan estas líneas el valor de *yn* debe ser cero, las siguientes iteraciones habrá de conservar su valor [13].

4.3.8.- Etapa de Zigzag inverso

Una vez que se ha decodificado el arreglo *zig* y se ha encontrado el coeficiente de *DC*, es preciso reordenar el arreglo unidimensional *zig* en un arreglo bidimensional, es decir el ordenamiento inverso al *Zigzag*. El algoritmo utilizado para la realización de este reordenamiento es el que se muestra en las ***Figuras 4.9***, ***4.10*** y ***4.11***, solamente que hay que intercambiar el orden en que se realizan las asignaciones es decir:

Para obtener un incremento de uno se tiene:

unidad_datos [j] [i = i + 1] = arreglo [k = k + 1];

Para obtener un incremento de siete:

unidad_datos [j = j + 1] [i = i - 1] = arreglo [k = k + 1];

El incremento de ocho se obtiene de la siguiente manera:

unidad_datos [j = j + 1] [i] = arreglo [k = k + 1];

Por último, el decremento de siete se logra mediante

unidad_datos [j = j - 1] [i = i + 1] = arreglo [k = k + 1];

La estructura de los algoritmos es exactamente la misma por lo tanto se omite su representación [13].

4.3.9.- Etapa de Decuantización

Una vez que se ha ordenado la *unidad de datos* a su posición original, los elementos de esta matriz deben multiplicarse con los elementos correspondientes de su matriz de cuantización, el proceso para obtener los coeficientes decuantizados debe cumplir con la ecuación **(4.1)**.

$$Matriz\_reconstruida(u,v) = \frac{Tabla\_cuantización(u,v) \times matriz\_cuantizada(u,v)}{Factor\ DIV} \qquad \textbf{(4.1)}$$

En este caso no es necesario utilizar las ***Tablas A.1*** y ***A.2***, sino que se utlizan las tablas originales de cuantización (***Sección 3.3.4.2***). Hay que señalar que los coeficientes reconstruidos se deben almacenar en un arreglo tipo FLOAT, la razón para efectuar esta acción es el hecho de que el siguiente proceso es la implementación de la *IDCT*, por tal motivo se utiliza este formato.

4.3.10.- Etapa IDCT

El siguiente paso en la decompresión de una *unidad de datos* es la aplicación de la *IDCT bidimensional*, para la realización de este proceso se ópto por utilizar el algoritmo propuesto por Wang, Suehiro y Hatori para la *IDCT unidimensional* que se muestra en la ***Figura 3.5*** [4][1].

Nuevamente al igual que en el caso de la ejecucción de la *DCT* es necesario aplicar la *IDCT* primero en el sentido horizontal, almacenando sus resultados en una matriz intermedia, a la cual se le aplica la *IDCT* en el sentido vertical (es decir por columnas), con lo cual se obtiene la *unidad de datos desplazada*.

4.3.11.- Etapa de corrimiento inverso

La última etapa para obtener las *unidades de datos* reconstruidas, es la aplicación de un corrimiento de nivel positivo sobre cada elemento de la unidad, pero también es necesario acotar los valores de dichos elementos pues es probable que aparezcan elementos cuyo resultado al momento de efectuar el corrimiento de nivel pueda superar el formato en los cuales se debe representar los datos reconstruidos *(unsigned char 0 a 255)*.

4.3.12.- Rutina para decodificar marcadores SOI y EOI

Como se hizo mención en el algoritmo de decompresión MJPEG, es preciso decodificar de los datos comprimidos los marcadores que indican el inicio y el final de cuadro, esto se realiza por medio del algoritmo de la ***Figura 4.27***.

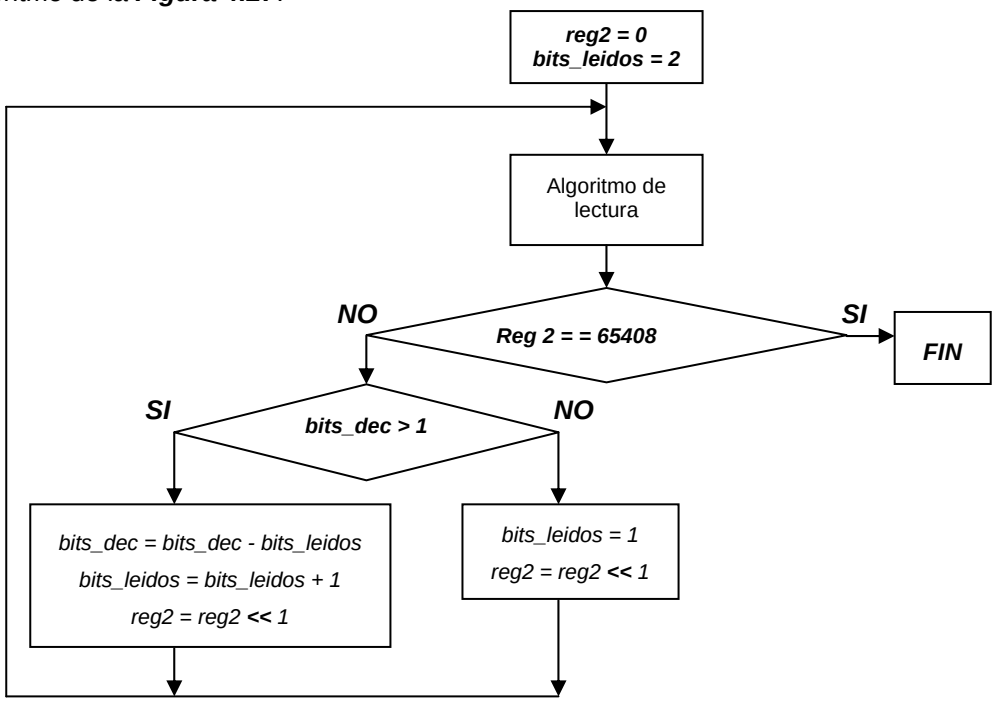


Figura 4.27.- Algoritmo para decodificar marcador SOI

De los dos bloques condicionales, el principal cumple con la sentencia, si la cadena de bits leída es igual al valor de 65408 (equivalente decimal para el código SOI), en caso de que el valor no sea el indicado se procede a realizar la lectura de otro bit, este proceso continua hasta que los bits leídos y el valor del código son iguales, cuando esta sentencia se cumple el algoritmo finaliza. Para poder decodificar el marcador EOI solo es necesario cambiar el valor de comparación, en este caso es 65409 [13].

4.3.13.- Asignación de las unidades de datos a memoria

Debido a que las *unidades de datos* se codificaron en un barrido secuencial – y en el caso que se procesan tres componentes se aplica la codificación entrelazada –, es preciso realizar la decodificación en igual sentido, tal como lo ilustra la ***Figura 4.5***, solo que ahora es necesario realizar la asignación de *unidades de datos* a los arreglos que han de almacenar la secuencia reconstruida, cuando finalizan todos los procesos de decodificación.

De está manera es como logramos obtener nuevamente la secuencia de video, se hace notar que esta secuencia difiere de la original y el grado en que difiere depende del valor del Factor *DIV*, el cual debe tener la misma magnitud para los procesos de compresión y decompresión; los resultados de aplicar los algoritmos descritos se muestran y evalúan en el siguiente capitulo.

RESUMEN

Este capítulo es la parte medular del presente trabajo de tesis, debido a esto la extensión del mismo. En él no solo se presentó una descripción de la herramienta principal de este sistema el DSP TMS320C6711, sino se planteó de manera detallada la realización tanto de la compresión como de la decompresión MJPEG. Si se desea analizar los programas finales se pueden consultar los anexos correspondientes ya que ahí se incluyen los programas completos.

Una vez descrita la realización del codec MJPEG, se procede a realizar las pruebas necesarias, esto con el fin de probar la validez de dichos algoritmos y el desarrollo de este trabajo.

CAPITULO V.- RESULTADOS OBTENIDOS

En este capitulo se presentan los resultados obtenidos de varías secuencias de video a las cuales se les aplicaron los algoritmos de Compresión y Decompresión MJPEG. Los principales criterios en la evaluación de estos resultados fueron sus factores de compresión alcanzados, la calidad visual de las secuencias reconstruidas, los tiempos de ejecución para los procesos de Compresión y Decocompresión, y los errores obtenidos al recuperar las secuencias.

5.1.- Evaluación de los niveles de compresión

En esta sección se presentan los resultados obtenidos para varias secuencias de video que fueron procesadas, este conjunto de secuencias está formado por un total de cuatro, de las cuales tres de ellas solo constan de su componente de Luminancia y la última consta de sus tres componentes (uno de Luminancia y dos de Crominancia).

La manera de evaluar los niveles de compresión alcanzados depende del factor *DIV* (***sección 3.3.5.2***), este factor nos permite modificar la cantidad de memoria que se almacenará en el arreglo *Datos_comprimidos.* El número total de localidades de memoria para dicho arreglo es contabilizado por medio de una variable que nos permite saber el tamaño alcanzado para los diferentes valores que toma el factor *DIV.* En las pruebas se utilizaron cuatro secuencias de video, cuyos resultados se muestran mas adelante.

Estas secuencias de video procesadas se encuentran en formato QCIF (176 píxeles x 144 Líneas), la razón para utilizar este tamaño es asumir un compromiso entre la cantidad de memoria utilizada para representar un cuadro y su calidad visual; por lo tanto se consideró que este formato cumple con ambos criterios. La secuencias procesadas constan de diferente resolución temporal, es decir, su número de cuadros es variable, tomando valores de 10, 13 o 20 cuadros.

Como se mencionó en la ***sección 4.2.7*** los datos comprimidos, se almacenan en el arreglo *Datos_comprimidos[contador]*, cuyas localidades de memoria constan de 16 bits, el total de localidades que ocupan los datos en este arreglo está dado por la variable *contador*, conociendo este valor es posible obtener el porcentaje de compresión alcanzado por los diferentes factores *DIV*, el procedimiento es el siguiente:

Debido a que cada localidad de memoria consta de 16 bits el total de bits en el arreglo *Datos_comprimidos* está dado por:

contador x 16 bits = total de bits en el arreglo

Un Byte consta de 8 bits por tal motivo el total de Bytes en el arreglo, se encuentra de la siguiente manera:

total de Bytes en el arreglo = total de bits en el arreglo / 8

Una vez que se encuentra el total de Bytes en el arreglo, el factor de compresión esta dado por la ecuación **(5.1)**.

%Factor de compresión = (Total de Bytes en el arreglo / Espacio original) x 100% **(5.1)**

Por ejemplo, consideremos que se comprimió una secuencia formada por 10 cuadros, que utiliza solo la componente de Luminancia por tal motivo su espacio original en memoria está dado por:

144 x 176 x 10 = 253,440 Bytes

La utilización del factor *DIV = 2* nos entrega que el valor de la variable *contador* es *11,875*, lo que implica que el número total de Bytes sea:

11,875 x 16 = 190,000 bits en el arreglo o *23,570 Bytes*

Por lo tanto el factor de compresión está dado por:

(23,750 Bytes / 253,440 Bytes) x 100 % = 9.371%

Este mismo procedimiento se realiza para los cinco valores que se le asignan al factor *DIV (2, 1, 0.5, 0.3 y 0.2)*, estos valores nos proveen diferentes grados de compresión, a su vez cada factor provee diferentes calidades visuales para las secuencias reconstruidas. El espacio original que ocupa cada secuencia, junto al espacio comprimido resultado del valor *DIV* utilizado, se muestran en las ***Tablas 5.1*** a ***5.4***, cada Tabla corresponde a una secuencia de video y en ella se logran apreciar sus correspondientes factores de compresión alcanzados. El primer resultado corresponde a la ***secuencia JEEP***, la cual está compuesta por 10 cuadros y solo utiliza su componente de Luminancia.

JEEP	***Espacio original***	***Espacio comprimido***	***Factor de compresión %***
Div = 2	***253,440 Bytes***	***23,750 Bytes***	***9.37%***
Div = 1	***253,440 Bytes***	***16,180 Bytes***	***6.38%***
Div = 0.5	***253,440 Bytes***	***10,680 Bytes***	***4.21%***
Div = 0.3	***253,440 Bytes***	***7,722 Bytes***	***3.05%***
Div = 0.2	***253,440 Bytes***	***6,076 Bytes***	***2.40%***

Tabla 5.1.- *Resultados obtenidos de la compresión para la* ***secuencia JEEP***

La ***Tabla 5.2*** corresponde a la ***secuencia CARA***, está formada por un total de 13 cuadros y solo utiliza la componente de Luminancia.

CARA	***Espacio original***	***Espacio comprimido***	***Factor de compresión %***
Div = 2	***329, 472 Bytes***	***41,458 Bytes***	***12.58%***
Div = 1	***329, 472 Bytes***	***28,024 Bytes***	***8.52%***
Div = 0.5	***329, 472 Bytes***	***19,024 Bytes***	***5.77%***
Div = 0.3	***329, 472 Bytes***	***14,040 Bytes***	***4.26%***
Div = 0.2	***329, 472 Bytes***	***11,164 Bytes***	***3.39%***

Tabla 5.2.- *Resultados obtenidos de la compresión para la* ***secuencia CARA***

La ***Tabla 5.3*** corresponde a la secuencia denominada BLOQUE, está formada por un total de 20 cuadros, y al igual que las dos anteriores solo utiliza la componente de Luminancia.

BLOQUE	***Espacio original***	***Espacio comprimido***	***Factor de compresión %***
Div = 2	***506, 880 Bytes***	***125,354 Bytes***	***24.37%***
Div = 1	***506, 880 Bytes***	***83,258 Bytes***	***16.42%***
Div = 0.5	***506, 880 Bytes***	***53,530 Bytes***	***10.56%***
Div = 0.3	***506, 880 Bytes***	***38,014 Bytes***	***7.49%***
Div = 0.2	***506, 880 Bytes***	***28,230 Bytes***	***5.56%***

Tabla 5.3.- *Resultados obtenidos de la compresión para la* ***secuencia BLOQUE***

La última secuencia de video procesada está formada por un total de 20 cuadros y utiliza sus tres componentes, el procedimiento para encontrar su factor de compresión es el mismo que se siguió en los casos anteriores, debido a que los datos comprimidos están entrelazados y se encuentran en un mismo arreglo. Los valores obtenidos se agrupan en la ***Tabla 5.4***.

COLOR	***Espacio original***	***Espacio comprimido***	***Factor de compresión %***
Div = 2	***1,520,640 Bytes***	***76,438 Bytes***	***5.03%***
Div = 1	***1,520,640 Bytes***	***49,456 Bytes***	***3.25%***
Div = 0.5	***1,520,640 Bytes***	***31,460 Bytes***	***2.07%***
Div = 0.3	***1,520,640 Bytes***	***23,322 Bytes***	***1.53%***
Div = 0.2	***1,520,640 Bytes***	***18,860 Bytes***	***1.24%***

Tabla 5.4.- *Resultados obtenidos de la compresión para la* ***secuencia COLOR***

Estas tablas proporcionan resultados muy variados, siendo el factor de compresión más bajo *24.37 %* y el más alto *1.24 %,* estas razones de compresión dependen principalmente de las escenas visuales que se están procesando. Además es importante recalcar que la secuencia que está compuesta por tres componentes, alcanza los factores de compresión mas altos, esto se debe a que las señales de crominancia se cuantizan más abruptamente, lo que genera que ocupen menos localidades de memoria en sus respectivos arreglos.

Los resultados obtenidos para el factor de compresión se muestran en la ***Figura 5.1***, el comportamiento de dichas curvas nos indican que conforme disminuye el valor del factor *DIV*, el porcentaje de compresión es más alto. Sin embargo hay que evaluar otros parámetros para el factor de mayor compresión (1.24%).

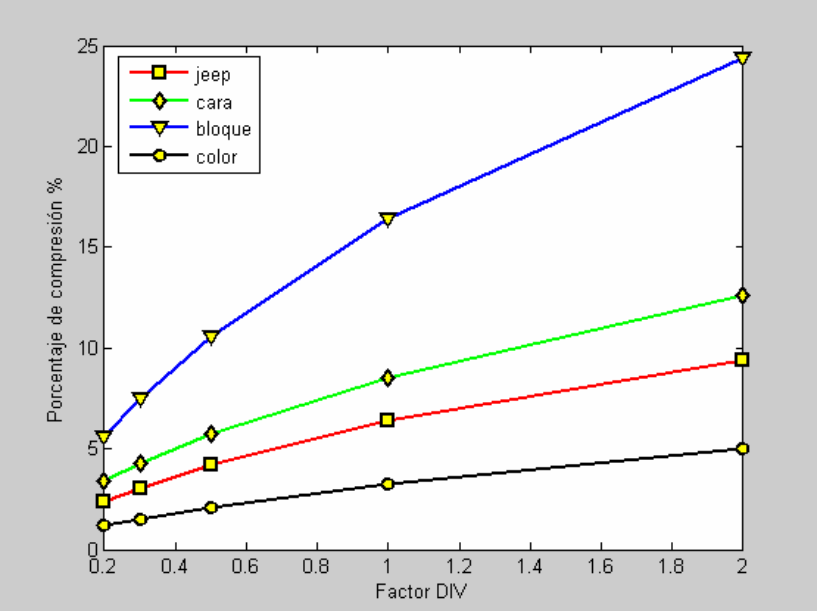


Figura 5.1.- Porcentaje de compresión contra factor DIV, para las cuatro secuencias procesadas

5.2.- Evaluación de los resultados visuales

El siguiente aspecto a tener en cuenta es el correspondiente a la calidad visual de la secuencia de video reconstruida. Los diferentes valores del factor *DIV* no solo modifican la cantidad de memoria que ocupan los arreglos, además tienen un efecto bastante significativo en la calidad visual de las secuencias reconstruidas. Por tal motivo a mayor grado de compresión, menor será la calidad visual de la secuencia reconstruida, debido a esto se presentan los resultados visuales obtenidos de aplicar los diferentes valores *DIV* para cada secuencia de video. Como las secuencias de video están formadas por un número diferente de cuadros, solo se mostrarán en forma impresa los cuadros inicial, intermedio y final de cada secuencia.

En las ***Figuras 5.2*** a ***5.25*** se realiza una comparación visual entre los cuadros de la secuencias originales y las secuencias comprimidas-decomprimidas en formato *MJPEG* a través del *DSP* por los diferentes valores de *DIV*, sin embargo una visualización total de la secuencia se logra a través del monitor de la PC, con ayuda del programa Matlab.

Figura 5.2.- Imágenes Inicial, intermedia y final de la ***secuencia JEEP*** original.

Figura 5.3.- Imágenes Inicial, intermedia y final de la ***secuencia JEEP*** reconstruida por el factor DIV=2.

Figura 5.4.- Imágenes Inicial, intermedia y final de la ***secuencia JEEP*** reconstruida por el factor DIV = 1.

Figura 5.5.- Imágenes Inicial, intermedia y final de la ***secuencia JEEP*** reconstruida por el factor DIV = 0.5.

Figura 5.6.- Imágenes Inicial, intermedia y final de la ***secuencia JEEP*** reconstruida por el factor DIV = 0.3.

Figura 5.7.- Imágenes Inicial, intermedia y final de la ***secuencia JEEP*** reconstruida por el factor DIV = 0.2.

Figura 5.8.- Imágenes Inicial, intermedia y final de la ***secuencia CARA*** original.

Figura 5.9.- Imágenes Inicial, intermedia y final de la ***secuencia CARA*** reconstruida por el factor DIV = 2.

Figura 5.10.- Imágenes Inicial, intermedia y final de la ***secuencia CARA*** reconstruida por el factor DIV = 1.

Figura 5.11.- Imágenes Inicial, intermedia y final de la ***secuencia CARA*** reconstruida por el factor DIV = 0.5.

Figura 5.12.- Imágenes Inicial, intermedia y final de la ***secuencia CARA*** reconstruida por el factor DIV = 0.3.

Figura 5.13.- Imágenes Inicial, intermedia y final de la ***secuencia CARA*** reconstruida por el factor DIV = 0.2.

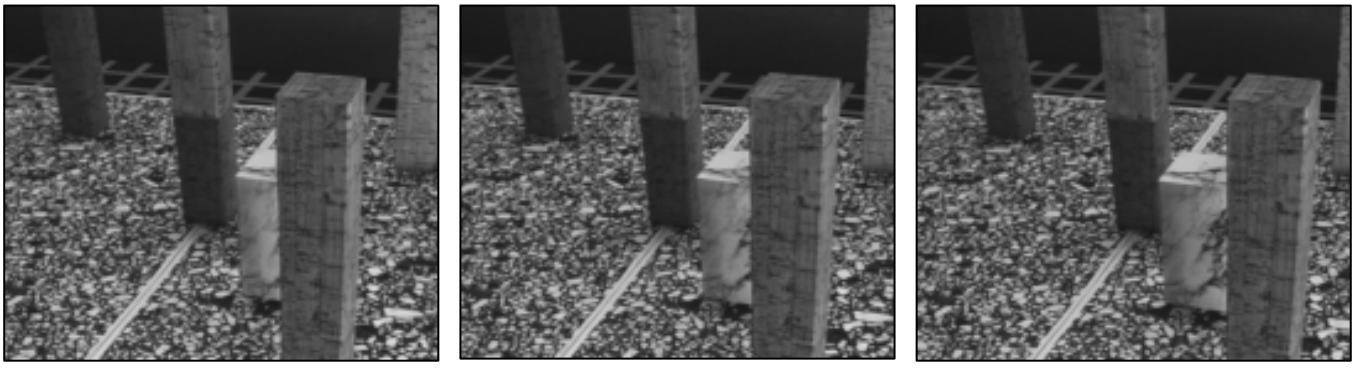


Figura 5.14.- Imágenes Inicial, intermedia y final de la ***secuencia BLOQUE*** original.

Figura 5.15.- Imágenes Inicial, intermedia y final de la ***secuencia BLOQUE*** reconstruida por el factor DIV = 2.

Figura 5.16.- Imágenes Inicial, intermedia y final de la ***secuencia BLOQUE*** reconstruida por el factor DIV = 1.

Figura 5.17.- Imágenes Inicial, intermedia y final de la ***secuencia BLOQUE*** reconstruida por el factor DIV = 0.5.

Figura 5.18.- Imágenes Inicial, intermedia y final de la ***secuencia BLOQUE*** reconstruida por el factor DIV = 0.3.

Figura 5.19.- Imágenes Inicial, intermedia y final de la ***secuencia BLOQUE*** reconstruida por el factor DIV = 0.2.

Figura 5.20.- Imágenes Inicial, intermedia y final de la ***secuencia COLOR*** original.

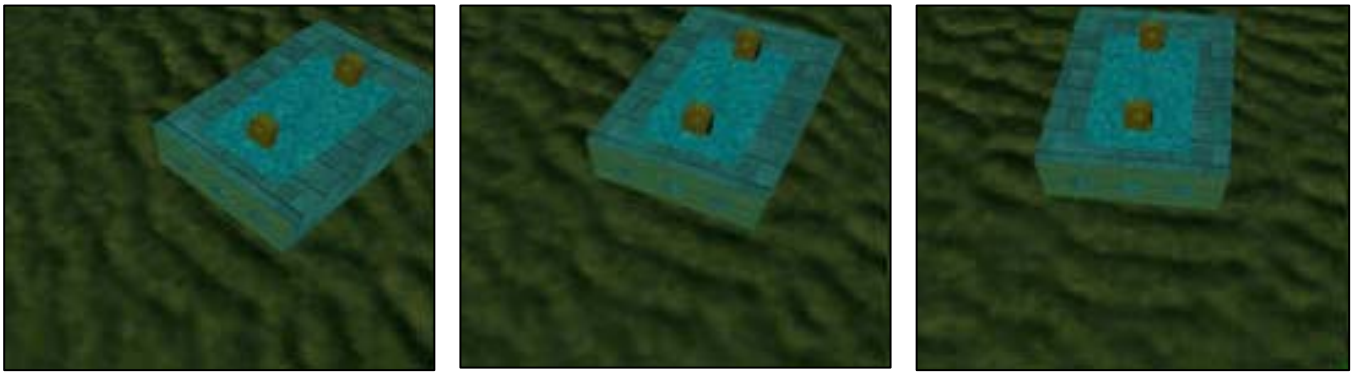


Figura 5.21.- Imágenes Inicial, intermedia y final de la ***secuencia COLOR*** reconstruida por el factor DIV = 2.

Figura 5.22.- Imágenes Inicial, intermedia y final de la ***secuencia COLOR*** reconstruida por el factor DIV = 1.

Figura 5.23.- Imágenes Inicial, intermedia y final de la ***secuencia COLOR*** reconstruida por el factor DIV = 0.5.

Figura 5.24.- Imágenes Inicial, intermedia y final de la ***secuencia COLOR*** reconstruida por el factor DIV = 0.3.

Figura 5.25.- Imágenes Inicial, intermedia y final de la ***secuencia COLOR*** reconstruida por el factor DIV = 0.2

Como se puede observar en las ***Figuras 5.2*** a ***5.25***, conforme el valor de *DIV* es menor, la calidad del video reconstruido disminuye, hasta tal punto que para un valor de 0.2 la calidad visual es muy mala, esto se debe a que factores *DIV* pequeños generan que los elementos de las matrices de cuantización tomen valores grandes con lo cual al momento de efectuar la etapa de cuantización, se pierden muchos de los coeficientes de *AC*, por tal motivo es difícil conservar los detalles espaciales finos.

Sin embargo, a pesar de que los resultados anteriores son bastantes descriptivos, es necesario realizar un análisis matemático del error que presentan las secuencias reconstruidas con respecto a las originales. El análisis realizado está basado en la ecuación **(5.2)**, la cual calcula la sumatoria total de la diferencia que se produce entre los respectivos elementos de la secuencia original y la reconstruida; en vista de que dicho número de elementos puede llegar a ser muy grande se optó por aplicar la ecuación **(5.2)** solo para el primer cuadro de cada secuencia. Los resultados obtenidos para este análisis se muestran en la ***Tabla 5.5***.

$$\%\,error = \left(\frac{1}{N} \sum_{i=1}^{N} \frac{| P_{original}(i) - P_{reconstruida}(i) |}{P_{original}(i)} \right) x\ 100 \qquad (5.2)$$

% error	JEEP	CARA	BLOQUE	COLOR
Div = 2	0.6942 %	3.33%	5.86%	1.52%
Div = 1	0.9590 %	4.57%	8.42%	2.22%
Div = 0.5	1.3567%	6.53%	11.00%	2.85%
Div = 0.3	1.8005%	8.80%	13.07%	4.05%
Div = 0.2	2.1426%	9.90%	15.43%	5.26%

Tabla 5.5.- Resultados obtenidos para el análisis de error en cada secuencia

En la ***Tabla 5.5*** y ***Figura 5.2.6*** se observa que conforme el factor *DIV* disminuye, el porcentaje de error se incrementa, esto se ve reflejado en la calidad visual obtenida para las secuencias reconstruidas (***Figuras 5.2*** a ***5.25***), puesto que a mayor porcentaje de error, las secuencias reconstruidas presentan menor calidad visual.

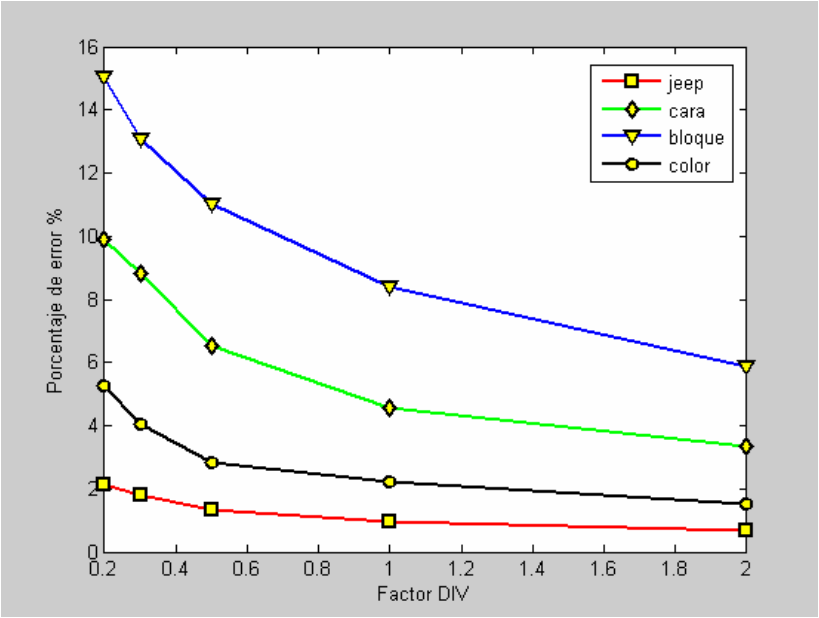


Figura 5.26.- Porcentaje de error contra factor DIV

Es interesante notar el hecho de que la ***secuencia BLOQUE*** presenta los porcentajes de error más altos, esto se debe a que visualmente esta secuencia es la más compleja, es decir, muchos de los píxeles que la forman están altamente decorrelacionados (tal como se puede ver en la parte del piso), por tal motivo al momento aplicar la ecuación de error **(5.2)**, se obtienen valores elevados, pues la imagen reconstruida difiere bastante de la original, - esto es difícil de apreciar a simple vista en la ***Figura 5.15***, la cual muestra las imágenes obtenidas para un *DIV = 2*, que nos permite apreciar una calidad visual bastante aceptable-.

Para finalizar con el análisis de error se muestran las *imágenes de error*, las cuales se obtuvieron para las tres componentes del primer cuadro de la ***secuencia COLOR***, para los valores de *DIV* = 2 y 0.2. Estas imágenes se obtienen del valor absoluto de las diferencias entre la imagen original y la reconstruida. Debido a que los valores resultantes de estas diferencias suelen ser pequeños se les aplica un offset de +128 con el fin de representar la imagen en tonos de gris, así mismo se modifica su contraste con el fin de tener una mejor calidad visual. En las ***Figuras 5.27*** y ***5.28***, se muestran las imágenes de error obtenidas para la componente de Luminancia, en estas dos figuras se aprecia como la imagen de error derivada de un valor de *DIV* pequeño conserva mas detalles espaciales pertenecientes a la imagen original, esto se debe a que existe una cantidad mayor de error.

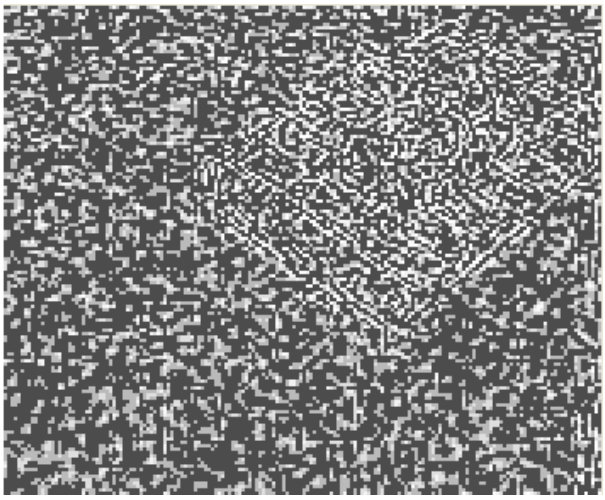


Figura 5.27.- Imagen de error para la componente de Luminancia Y, obtenida para el valor de DIV = 2

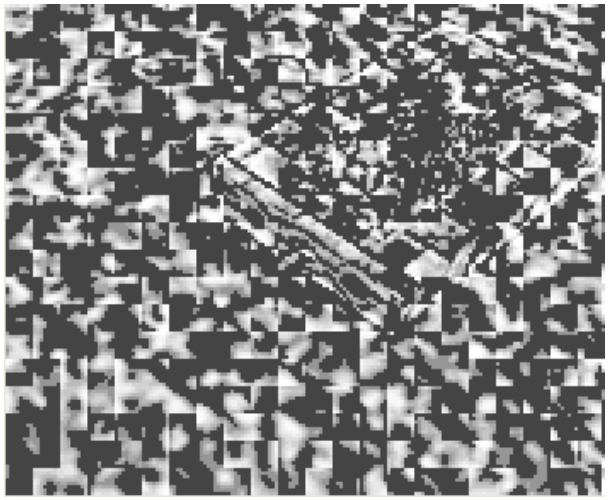


Figura 5.28.- Imagen de error para la componente de Luminancia Y, obtenida para el valor de DIV = 0.2

En las ***Figuras 5.29*** y ***5.30***, se muestran las *imágenes de error* pertenecientes a la componente de Crominancia Cr, en la ***Figura 5.30***, se logra apreciar como la imagen resultante contiene una gran cantidad de detalles espaciales pertenecientes a la imagen original lo cual explica el hecho de que las imágenes de la ***Figura 5.25***, presenten una calidad visual muy mala, ya que gran parte de su información se encuentra contenida en las *imágenes de error.*

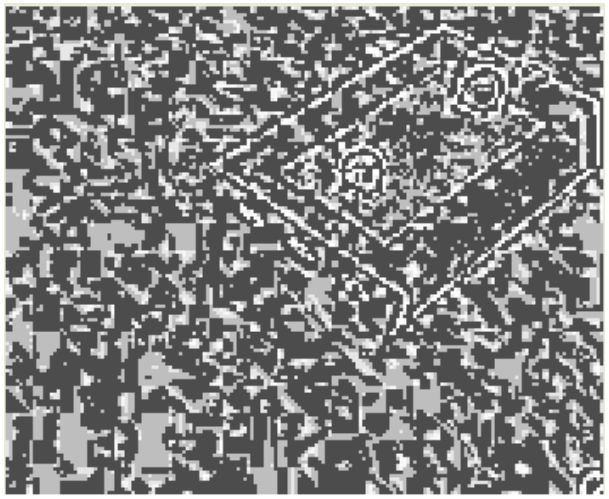


Figura 5.29.- Imagen de error para la componente de Crominancia Cr, obtenida para el valor de DIV = 2

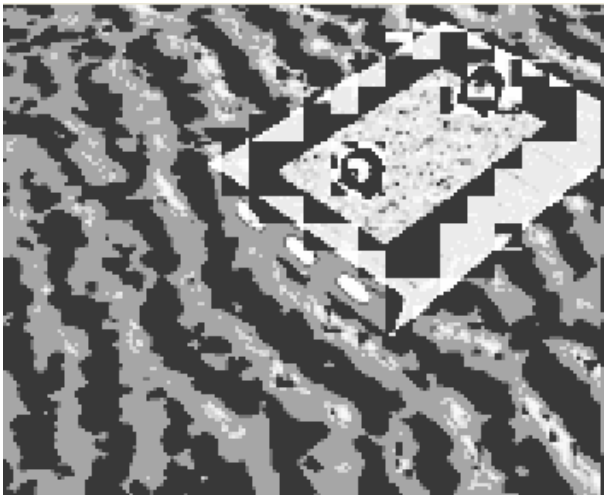


Figura 5.30.- Imagen de error para la componente de Crominancia Cr, obtenida para el valor de DIV = 0.2

Las últimas imágenes de error pertenecen a la componente de crominancia Cb, las cuales se muestran en las ***Figuras 5.31*** y ***5.32***, en ellas nuevamente se aprecia que la *imagen de error* generada por el valor más bajo de *DIV* conserva una cantidad mayor de detalles espaciales con respecto a la *imagen de error* que es generada por el valor de *DIV* mayor.

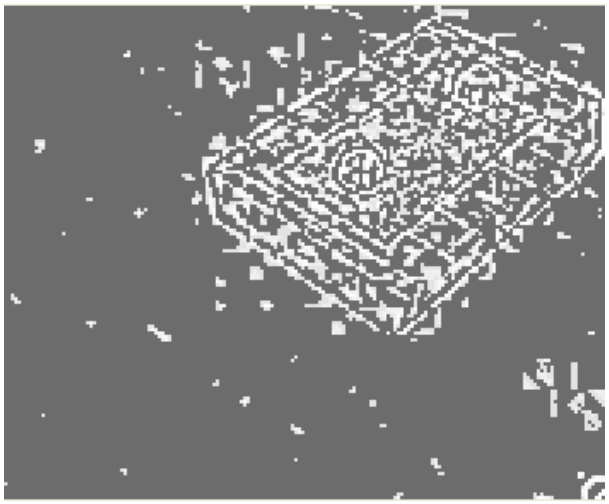


Figura 5.31.- Imagen de error para la componente de Crominancia Cb, obtenida para el valor de DIV = 2

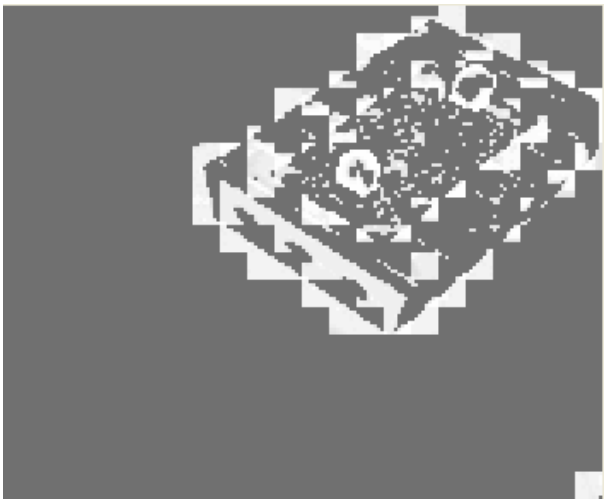


Figura 5.32.- Imagen de error para la componente de Crominancia Cb, obtenida para el valor de DIV = 0.2

5.3.- Evaluación de los tiempos de ejecución

El último parámetro importante a evaluar, es el tiempo de ejecución tanto para los algoritmos de compresión como para los de decompresión utilizando una arquitectura *DSP*. Los tiempos de ejecución que se obtuvieron fueron para los diferentes valores de *DIV* que se aplicaron a las distintas secuencias, estos tiempos son muy variados porque dependen principalmente del contenido visual de la escena, así como del número de cuadros que se han de procesar.

Los resultados se muestran en las ***Tablas 5.6*** a ***5.9***, y se obtuvieron gracias al ambiente integral *CCS*, estos tiempos tienen como base el nivel máximo de optimización de código para el *DSP 6711* que trabaja a 1200 Millones de instrucciones por segundo (MIPS) a una frecuencia de 150 MHz [21][24].

JEEP	***Compresión***	***Decompresión***
Div = 2	***1.6413 seg***	***3.169 seg***
Div = 1	***1.4802 seg***	***2.610 seg***
Div = 0.5	***1.3742 seg***	***2.193 seg***
Div = 0.3	***1.3172 seg***	***2.012 seg***
Div = 0.2	***1.2886 seg***	***1.816 seg***

Tabla 5.6.- *Tiempos de ejecución para la* ***secuencia JEEP***, que consta de 10 cuadros

CARA	***Compresión***	***Decompresión***
Div = 2	***2.3395 seg***	***4.9454 seg***
Div = 1	***2.0727 seg***	***3.9312 seg***
Div = 0.5	***1.8975 seg***	***3.2430 seg***
Div = 0.3	***1.7991 seg***	***2.8874 seg***
Div = 0.2	***1.7419 seg***	***2.6066 seg***

Tabla 5.7.- *Tiempos de ejecución para la* ***secuencia CARA***, que consta de 13 cuadros

BLOQUE	***Compresión***	***Decompresión***
Div = 2	***4.8689 seg***	***12.2561 seg***
Div = 1	***4.0028 seg***	***9.1435 seg***
Div = 0.5	***3.4007 seg***	***6.9445 seg***
Div = 0.3	***3.0668 seg***	***5.8862 seg***
Div = 0.2	***2.8610 seg***	***5.1454 seg***

Tabla 5.8.- *Tiempos de ejecución para la* ***secuencia BLOQUE***, que consta de 20 cuadros

COLOR	***Compresión***	***Decompresión***
Div = 2	***8.9599 seg***	***13.706 seg***
Div = 1	***8.4321 seg***	***11.7485 seg***
Div = 0.5	***8.0897 seg***	***10.4241 seg***
Div = 0.3	***7.9332 seg***	***9.3466 seg***
Div = 0.2	***7.8427 seg***	***8.9832 seg***

Tabla 5.9.- *Tiempos de ejecución para la* ***secuencia COLOR***, que consta de 20 cuadros

Los resultados de las tablas anteriores se muestran gráficamente en las ***Figuras 5.33*** y ***5.34***, al analizar estos resultados se observa que los tiempos de ejecución para la Decompresión en todas las secuencias son mayores que los tiempos correspondientes para la etapa de Compresión; esto se debe a que al momento de decodificar los códigos Huffman el algoritmo realiza diversas búsquedas

hasta que encuentra con el código correspondiente, así mismo el cálculo de la *IDCT* es alrededor de 2.64 veces mayor que el de la *DCT*, debido a estos motivos es que los tiempos se incrementan.

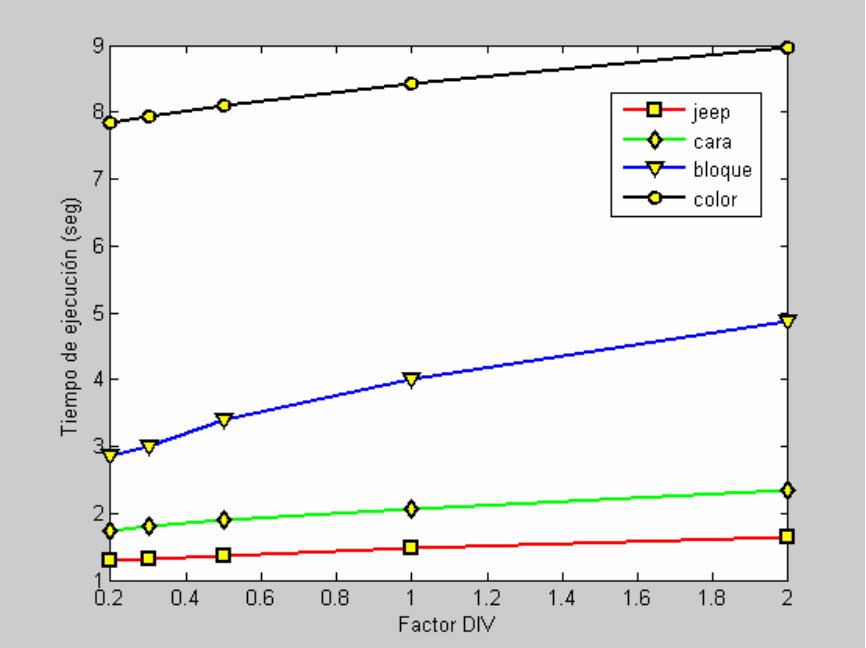


Figura 5.33.- Tiempo de ejecución para la Compresión contra factor DIV

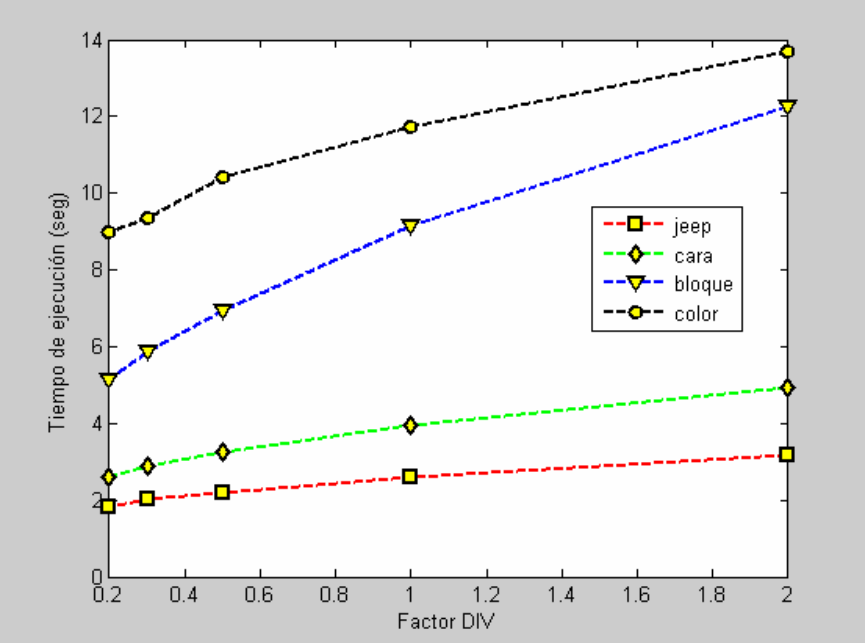


Figura 5.34.- Tiempo de ejecución para la Decompresión contra factor DIV

En la ***Figura 5.34***, se observa que de las tres secuencias que solo cuentan con la componente de luminancia, la que toma más tiempo para la ejecución del algoritmo de decompresión es la ***secuencia BLOQUE***, esto se debe a que el arreglo comprimido cuenta con más información, por tal motivo la búsqueda de sus códigos Huffman es más larga.

Si se comparan los tiempos de ejecución del algoritmo de compresión entre las ***secuencias BLOQUE*** y ***COLOR*** (que contienen el mismo número de cuadros), se observa que los tiempos de la ***secuencia COLOR*** son casi el doble que los de la ***secuencia BLOQUE***; esto se debe a que la ***secuencia COLOR*** contiene tres componentes.

5.4.- Evaluación de tiempos en el DSP TMS320C6713

Debido a que los tiempos obtenidos en la ejecución de los algoritmos de compresión y decompresión en el *DSP* TMS320C6711, son inadecuados para poder realizar una implementación en tiempo real, se procedió a efectuar la ejecución de dichos algoritmos en el *DSP* TMS320C6713 el cual puede realizar hasta 1800 MIPS a una frecuencia de reloj de 225 MHz [21][24]. Los resultados obtenidos se aplicaron a todas las secuencias solo para el valor *DIV =2*, esto se debe a que dicho valor nos provee los tiempos de ejecución más grandes y las mejores calidades visuales, los valores obtenidos se muestran en la ***Tabla 5.10***.

SECUENCIA	Compresión	Decompresión
1.- JEEP	0.6863 seg	1.1504 seg
2.- CARA	0.93443 seg	1.8384 seg
3.- BLOQUE	2.005 seg	4.53 seg
4.- COLOR	3.3093 seg	5.0951 seg

Tabla 5.10.- *Tiempos de ejecución obtenidos en el DSP TMS320C6713*

Los desempeños de ambos *DSP´s*, se comparan en la ***Figura 5.35*** para sus respectivos valores de *DIV*, en estas gráficas podemos observar como los tiempos de ejecución se mejoran considerablemente por el C6713 (aproximadamente por un factor de 2.4), tanto para el proceso de compresión como para el de decompresión. Además los tiempos obtenidos para las dos primeras secuencias son factibles para implementarse en tiempo real.

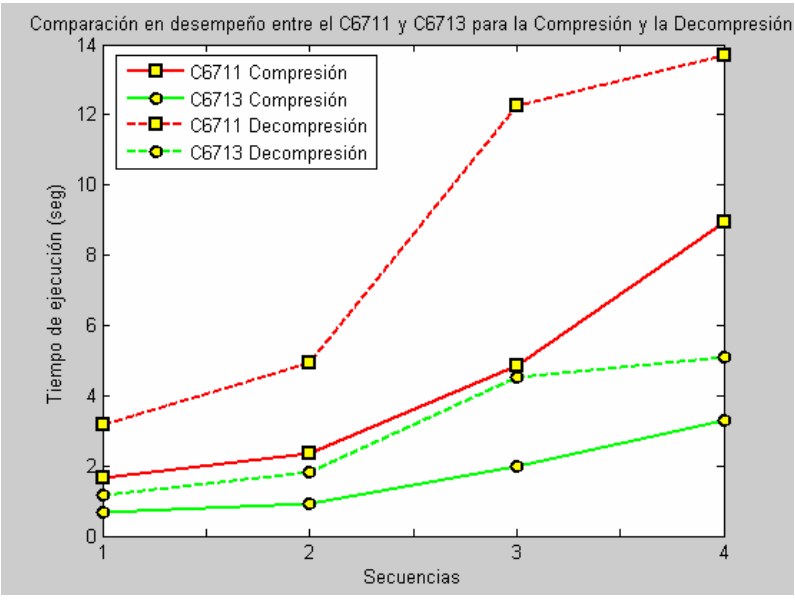


Figura 5.35.- Comparación en desempeño entre el C6711 y el C6713

Para concluir con el análisis de los tiempos de ejecución, se procedió a determinar el tiempo que se ocupa para la realización una *DCT* e *IDCT bidimensionales* en una *unidad de datos*, la evaluación de estos parámetros se debe a que dichos procesos son los que necesitan mayor poder de cómputo en la codificación y decodificación de un bloque de 8x8. Nuevamente los valores resultantes se comparan entre el C6711 y C6713, con el fin de observar su desempeño. Los valores obtenidos se muestran en la ***Tabla 5.11***.

		Ciclos	***Tiempo de ejecución***
TMS320C6711	***DCT bidimensional (Wang- Suehiro- Hatori)***	***9,883***	***65.886 μ seg***
	IDCT bidimensional (Wang- Suehiro- Hatori)	***26,143***	***176.0866 μ seg***
TMS320C6713	***DCT bidimensional (Wang- Suehiro- Hatori)***	***7,158***	***31.813 μ seg***
	IDCT bidimensional (Wang- Suehiro- Hatori)	***9,937***	***44.164 μ seg***

Tabla 5.11.- *Comparación de tiempos de ejecución para una unidad de datos*

En la tabla anterior se observa como los tiempos de ejecución mejoran cuando se utiliza el C6713, sobretodo para la *IDCT*, también es posible observar como el tiempo de ejecución casi se triplica entre la *DCT* y la *IDCT* al evaluar el C6711, esto se debe a que la *IDCT* tiene como operandos valores fraccionarios los cuales necesariamente deben estar en formato tipo FLOAT y por tanto incrementan su tiempo de ejecución. De la ***Tabla 5.11*** se observa que utilizando el *DSP* C6713 la diferencia entre la *DCT* e *IDCT*, no es tan grande como en el caso del C6711, esto se debe a que el C6713 puede manejar un numero mayor de millones de operaciones en punto flotante por segundo (MFLOPS) 1350 MFLOPS, en tanto el C6711 solo maneja 600 MFLOPS, estos resultados demuestran que el C6713 tiene un desempeño más alto en la implementación de los algoritmos *MJPEG*. En la actualidad existen arquitecturas como el TMS320C6416 que opera a 1GHz pudiendo ejecutar 8000 MIPS, tal desempeño nos permite no solo la implementación en tiempo real, sino el manejar resoluciones espaciales y temporales mayores [24].

RESUMEN

En este capitulo se presentó el análisis de la información correspondiente a todos los resultados que se obtuvieron en este trabajo, como se pudo observar los valores que se presentan para los diferentes parámetros evaluados de las secuencias son muy diversos. Lo que se puede concluir de ellos es que generalmente los tiempos de compresión son mucho menores, que los que nos entrega el algoritmo de decompresión típicamente por un factor de ***2.6*** *en el C6711 y* ***1.38*** *en el C6713. Además cuando se procesa una secuencia de vídeo con las tres componentes, los factores de compresión alcanzados son mas altos que cuando solo se procesa la componente de Luminancia. Por último se consideró que para lograr un equilibrio entre el tiempo de ejecución, el factor de compresión y la calidad del vídeo reconstruido el valor de DIV mas adecuado es 0.5 ya que proporciona desempeños bastantes aceptables para todos los parámetros evaluados.*

CONCLUSIONES

Como se observó en el ***Capítulo III***, la parte fundamental para la implementación del codificador-decodificador *MJPEG* es la Transformada Coseno Discreta (*DCT*) y su Transformada Inversa (*IDCT*); estas transformadas matemáticas requieren de 4096 operaciones MAC (Multiplicación-Acumulación), utilizando el criterio de la funciones base para el procesamiento de una *unidad de datos de 8x8,* debido a esto se optó por utilizar un *Procesador Digital de Señales* (*DSP*).

Los parámetros evaluados (factor de compresión, calidad visual y tiempos de ejecución) nos permiten formar el siguiente criterio, a mayor porcentaje de compresión, menor será la calidad visual de la secuencias reconstruidas, tal como lo comprueban los resultados de error obtenidos, de hecho estos resultados nos indican que entre más alta sea la complejidad espacial de una escena, mayor será el error de las secuencias reconstruidas, como se observó en el procesamiento de la ***secuencia BLOQUE***, y por tanto se reducirá el tiempo de ejecución tanto para las etapas de compresión, como de decompresión.

Para lograr un equilibrio entre el factor de compresión y la calidad visual de acuerdo a nuestros resultados, consideramos que lo más adecuado es utilizar el valor de 0.5 para *DIV*; pues este nos entrega calidades visuales bastante aceptables así como factores de compresión entre el 2 y 10% aproximadamente. Así mismo se encontró que cuando se codifican una secuencia de video que consta de sus tres componentes los factores de compresión alcanzados son mas elevados, que cuando solo se procesa una sola componente; esto se debe a que las dos componentes de Crominancia se codifican mas abruptamente que la señal de Luminancia.

Como se mostró en el ***Capítulo V***, los tiempos de decompresión son mayores a los tiempos de compresión, esto se debe a que en la etapa de decompresión se deben realizar diversas comparaciones hasta encontrar el código Huffman correcto. Además como se observó de los resultados de la ***Tabla 5.11***, los tiempos de ejecución para la *IDCT* son más grandes que para los de la *DCT*, este es otro factor para aumentar los tiempos de decompresión.

Los tiempos de ejecución nos permiten establecer que si bien es cierto que para una secuencia que está formada por 20 cuadros y con sus tres componentes los tiempos de compresión fluctúan entre 7.8 y 9 seg, los de decompresión están entre 9 y 14 seg. Casi es posible realizar una implementación del sistema en tiempo real. Para lograr reducir estos tiempos de ejecución se puede optar por optimizar al máximo el código fuente del programa (de ser preciso realizar la implementación de la *DCT* e *IDCT* en lenguaje ensamblador para eficientar el desempeño) y utilizar un *DSP* que trabaje con un número mayor de MIPS, tal como es el caso del C713 u otro mas potente, logrando mejorar los tiempos de ejecución para la compresión y la decompresión alrededor de un factor del 2.4.

El hecho de poder utilizar un *DSP* que trabaje a un número mayor de ciclos de reloj y logre desempeñar una mayor cantidad de MIPS, no solo hace factible la implementación del sistema en tiempo real, sino también nos permitirá aumentar la resolución espacial de las secuencias de video a procesar, obteniendo mejores calidades visuales.

Una de las futuras aplicaciones prácticas para este trabajo podría ser la creación de un *sistema de vigilancia de una puerta* o una *caseta pública para videollamada*, la parte emisora que realizaría la compresión puede estar formada por el *DSP* y una tarjeta de adquisición de video; una vez obtenido los datos comprimidos estos pueden viajar a través de una red IP y para eficientar el desempeño del sistema la parte receptora estará formada por una computadora que realizará la decompresión permitiendo desplegar el video con ayuda de su monitor.

ANEXO A.-

Tablas requeridas por el Codificador y Decodificador modo básico MJPEG

0.0625	*0.0909*	*0.1000*	*0.0625*	*0.0417*	*0.0250*	*0.0196*	*0.0164*
0.0833	*0.0833*	*0.0714*	*0.0526*	*0.0385*	*0.0172*	*0.0167*	*0.0182*
0.0714	*0.0769*	*0.0625*	*0.0417*	*0.0250*	*0.0175*	*0.0145*	*0.0179*
0.0714	*0.0588*	*0.0455*	*0.0345*	*0.0196*	*0.0115*	*0.0125*	*0.0161*
0.0556	*0.0455*	*0.0270*	*0.0179*	*0.0147*	*0.0092*	*0.0097*	*0.0130*
0.0417	*0.0286*	*0.0182*	*0.0156*	*0.0123*	*0.0096*	*0.0088*	*0.0109*
0.0204	*0.0156*	*0.0128*	*0.0115*	*0.0097*	*0.0083*	*0.0083*	*0.0099*
0.0139	*0.0109*	*0.0105*	*0.0102*	*0.0089*	*0.0100*	*0.0097*	*0.0101*

Tabla A.1*.- Tabla de cuantización de valores inversos para la Luminancia*

0.0588	*0.0555*	*0.0416*	*0.0212*	*0.0101*	*0.0101*	*0.0101*	*0.0101*
0.0555	*0.0476*	*0.0384*	*0.0151*	*0.0101*	*0.0101*	*0.0101*	*0.0101*
0.0416	*0.0384*	*0.0178*	*0.0101*	*0.0101*	*0.0101*	*0.0101*	*0.0101*
0.0212	*0.0151*	*0.0101*	*0.0101*	*0.0101*	*0.0101*	*0.0101*	*0.0101*
0.0101	*0.0101*	*0.0101*	*0.0101*	*0.0101*	*0.0101*	*0.0101*	*0.0101*
0.0101	*0.0101*	*0.0101*	*0.0101*	*0.0101*	*0.0101*	*0.0101*	*0.0101*
0.0101	*0.0101*	*0.0101*	*0.0101*	*0.0101*	*0.0101*	*0.0101*	*0.0101*
0.0101	*0.0101*	*0.0101*	*0.0101*	*0.0101*	*0.0101*	*0.0101*	*0.0101*

Tabla A.2.- *Tabla de cuantización de valores inversos para la Crominancia*

Categoría SSSS	*Longitud del Código*	*Código Huffman*
0	*2*	***00***
1	*3*	***010***
2	*3*	***011***
3	*3*	***100***
4	*3*	***101***
5	*3*	***110***
6	*4*	***1110***
7	*5*	***11110***
8	*6*	***111110***
9	*7*	***1111110***
10	*8*	***11111110***
11	*9*	***111111110***

Tabla A.3.- *Códigos base para la generación de diferencias DPCM señal Luminancia*

Categoría SSSS	*Longitud del Código*	*Código Huffman*
0	*2*	***00***
1	*2*	***01***
2	*2*	***10***
3	*3*	***110***
4	*4*	***1110***
5	*5*	***11110***
6	*6*	***111110***
7	*7*	***1111110***
8	*8*	***11111110***
9	*9*	***111111110***
10	*10*	***1111111110***
11	*11*	***11111111110***

Tabla A.4.- *Códigos base para la generación de diferencias DPCM señal Crominancia*

Cadena Categoría	Longitud del código	Código Huffman	Cadena Categoría	Longitud del código	Código Huffman
0/0 (EOB)	4	**1010**			
0/1	2	**00**	**4/1**	6	**111011**
0/2	2	**01**	**4/2**	10	**1111111000**
0/3	3	**100**	**4/3**	16	**1111111110010110**
0/4	4	**1011**	**4/4**	16	**1111111110010111**
0/5	5	**11010**	**4/5**	16	**1111111110011000**
0/6	7	**1111000**	**4/6**	16	**1111111110011001**
0/7	8	**11111000**	**4/7**	16	**1111111110011010**
0/8	10	**1111110110**	**4/8**	16	**1111111110011011**
0/9	16	**1111111110000010**	**4/9**	16	**1111111110011100**
0/A	16	**1111111110000011**	**4/A**	16	**1111111110011101**
1/1	4	**1100**	**5/1**	7	**1111010**
1/2	5	**11011**	**5/2**	11	**11111110111**
1/3	7	**1111001**	**5/3**	16	**1111111110011110**
1/4	9	**111110110**	**5/4**	16	**1111111110011111**
1/5	11	**11111110110**	**5/5**	16	**1111111110100000**
1/6	16	**1111111110000100**	**5/6**	16	**1111111110100001**
1/7	16	**1111111110000101**	**5/7**	16	**1111111110100010**
1/8	16	**1111111110000110**	**5/8**	16	**1111111110100011**
1/9	16	**1111111110000111**	**5/9**	16	**1111111110100100**
1/A	16	**1111111110001000**	**5/A**	16	**1111111110100101**
2/1	5	**11100**	**6/1**	7	**1111011**
2/2	8	**11111001**	**6/2**	12	**111111110110**
2/3	10	**1111110111**	**6/3**	16	**1111111110100110**
2/4	12	**111111110100**	**6/4**	16	**1111111110100111**
2/5	16	**1111111110001001**	**6/5**	16	**1111111110101000**
2/6	16	**1111111110001010**	**6/6**	16	**1111111110101001**
2/7	16	**1111111110001011**	**6/7**	16	**1111111110101010**
2/8	16	**1111111110001100**	**6/8**	16	**1111111110101011**
2/9	16	**1111111110001101**	**6/9**	16	**1111111110101100**
2/A	16	**1111111110001110**	**6/A**	16	**1111111110101101**
3/1	6	**111010**	**7/1**	8	**11111010**
3/2	9	**111110111**	**7/2**	12	**111111110111**
3/3	12	**111111110101**	**7/3**	16	**1111111110101110**
3/4	16	**1111111110001111**	**7/4**	16	**1111111110101111**
3/5	16	**1111111110010000**	**7/5**	16	**1111111110110000**
3/6	16	**1111111110010001**	**7/6**	16	**1111111110110001**
3/7	16	**1111111110010010**	**7/7**	16	**1111111110110010**
3/8	16	**1111111110010011**	**7/8**	16	**1111111110110011**
3/9	16	**1111111110010100**	**7/9**	16	**1111111110110100**
3/A	16	**1111111110010101**	**7/A**	16	**1111111110110101**

Cadena Categoría	Longitud del código	Código Huffman	Cadena Categoría	Longitud del código	Código Huffman
8/1	9	**111111000**	**C/1**	10	**1111111010**
8/2	15	**111111111000000**	**C/2**	16	**1111111111011001**
8/3	16	**1111111110110110**	**C/3**	16	**1111111111011010**
8/4	16	**1111111110110111**	**C/4**	16	**1111111111011011**
8/5	16	**1111111110111000**	**C/5**	16	**1111111111011100**
8/6	16	**1111111110111001**	**C/6**	16	**1111111111011101**
8/7	16	**1111111110111010**	**C/7**	16	**1111111111011110**
8/8	16	**1111111110111011**	**C/8**	16	**1111111111011111**
8/9	16	**1111111110111100**	**C/9**	16	**1111111111100000**
8/A	16	**1111111110111101**	**C/A**	16	**1111111111100001**
9/1	9	**111111001**	**D/1**	11	**11111111000**
9/2	16	**1111111110111110**	**D/2**	16	**1111111111100010**
9/3	16	**1111111110111111**	**D/3**	16	**1111111111100011**
9/4	16	**1111111111000000**	**D/4**	16	**1111111111100100**
9/5	16	**1111111111000001**	**D/5**	16	**1111111111100101**
9/6	16	**1111111111000010**	**D/6**	16	**1111111111100110**
9/7	16	**1111111111000011**	**D/7**	16	**1111111111100111**
9/8	16	**1111111111000100**	**D/8**	16	**1111111111101000**
9/9	16	**1111111111000101**	**D/9**	16	**1111111111101001**
9/A	16	**1111111111000110**	**D/A**	16	**1111111111101010**
A/1	9	**111111010**	**E/1**	16	**1111111111101011**
A/2	16	**1111111111000111**	**E/2**	16	**1111111111101100**
A/3	16	**1111111111001000**	**E/3**	16	**1111111111101101**
A/4	16	**1111111111001001**	**E/4**	16	**1111111111101110**
A/5	16	**1111111111001010**	**E/5**	16	**1111111111101101**
A/6	16	**1111111111001011**	**E/6**	16	**1111111111110000**
A/7	16	**1111111111001100**	**E/7**	16	**1111111111110001**
A/8	16	**1111111111001101**	**E/8**	16	**1111111111110010**
A/9	16	**1111111111001110**	**E/9**	16	**1111111111110011**
A/A	16	**1111111111001111**	**E/A**	16	**1111111111110100**
B/1	10	**1111111001**	**F/1**	16	**1111111111110101**
B/2	16	**1111111111010000**	**F/2**	16	**1111111111110110**
B/3	16	**1111111111010001**	**F/3**	16	**1111111111110111**
B/4	16	**1111111111010010**	**F/4**	16	**1111111111111000**
B/5	16	**1111111111010011**	**F/5**	16	**1111111111111001**
B/6	16	**1111111111010100**	**F/6**	16	**1111111111111010**
B/7	16	**1111111111010101**	**F/7**	16	**1111111111111011**
B/8	16	**1111111111010110**	**F/8**	16	**1111111111111100**
B/9	16	**1111111111010111**	**F/9**	16	**1111111111111101**
B/A	16	**1111111111011000**	**F/A**	16	**1111111111111110**
			F/0 (ZRL)	11	**11111111001**

Tabla A.5.- *Códigos base para la generación de coeficientes AC señal Luminancia*

Cadena Categoría	Longitud del código	Código Huffman	Cadena Categoría	Longitud del código	Código Huffman
0/0 (EOB)	2	**00**			
0/1	2	**01**	**4/1**	6	**111010**
0/2	3	**100**	**4/2**	9	**111110110**
0/3	4	**1010**	**4/3**	16	**1111111110010111**
0/4	5	**11000**	**4/4**	16	**1111111110011000**
0/5	5	**11001**	**4/5**	16	**1111111110011001**
0/6	6	**111000**	**4/6**	16	**1111111110011010**
0/7	7	**1111000**	**4/7**	16	**1111111110011011**
0/8	9	**111110100**	**4/8**	16	**1111111110011100**
0/9	10	**1111110110**	**4/9**	16	**1111111110011101**
0/A	12	**111111110100**	**4/A**	16	**1111111110011110**
1/1	4	**1011**	**5/1**	6	**111011**
1/2	6	**111001**	**5/2**	10	**1111111001**
1/3	8	**11110110**	**5/3**	16	**1111111110011111**
1/4	9	**111110101**	**5/4**	16	**1111111110100000**
1/5	11	**11111110110**	**5/5**	16	**1111111110100001**
1/6	12	**111111110101**	**5/6**	16	**1111111110100010**
1/7	16	**1111111110001000**	**5/7**	16	**1111111110100011**
1/8	16	**1111111110001001**	**5/8**	16	**1111111110100100**
1/9	16	**1111111110001010**	**5/9**	16	**1111111110100101**
1/A	16	**1111111110001011**	**5/A**	16	**1111111110100110**
2/1	5	**11010**	**6/1**	7	**1111001**
2/2	8	**11110111**	**6/2**	11	**11111110111**
2/3	10	**1111110111**	**6/3**	16	**1111111110100111**
2/4	12	**111111110110**	**6/4**	16	**1111111110101000**
2/5	15	**111111111000010**	**6/5**	16	**1111111110101001**
2/6	16	**1111111110001100**	**6/6**	16	**1111111110101010**
2/7	16	**1111111110001101**	**6/7**	16	**1111111110101011**
2/8	16	**1111111110001110**	**6/8**	16	**1111111110101100**
2/9	16	**1111111110001111**	**6/9**	16	**1111111110101101**
2/A	16	**1111111110010000**	**6/A**	16	**1111111110101110**
3/1	5	**11011**	**7/1**	7	**1111010**
3/2	8	**11111000**	**7/2**	11	**11111111000**
3/3	10	**1111111000**	**7/3**	16	**1111111110101111**
3/4	12	**111111110111**	**7/4**	16	**1111111110110000**
3/5	16	**1111111110010001**	**7/5**	16	**1111111110110001**
3/6	16	**1111111110010010**	**7/6**	16	**1111111110110010**
3/7	16	**1111111110010011**	**7/7**	16	**1111111110110011**
3/8	16	**1111111110010100**	**7/8**	16	**1111111110110100**
3/9	16	**1111111110010101**	**7/9**	16	**1111111110110101**
3/A	16	**1111111110010110**	**7/A**	16	**1111111110110110**

Cadena Categoría	Longitud del código	Código Huffman	Cadena Categoría	Longitud del código	Código Huffman
8/1	8	**11111001**	**C/1**	9	**111111010**
8/2	16	**1111111110110111**	**C/2**	16	**1111111111011011**
8/3	16	**1111111110111000**	**C/3**	16	**1111111111011100**
8/4	16	**1111111110111001**	**C/4**	16	**1111111111011101**
8/5	16	**1111111110111010**	**C/5**	16	**1111111111011110**
8/6	16	**1111111110111011**	**C/6**	16	**1111111111011111**
8/7	16	**1111111110111100**	**C/7**	16	**1111111111100000**
8/8	16	**1111111110111101**	**C/8**	16	**1111111111100001**
8/9	16	**1111111110111110**	**C/9**	16	**1111111111100010**
8/A	16	**1111111110111111**	**C/A**	16	**1111111111100011**
9/1	9	**111110111**	**D/1**	11	**11111111001**
9/2	16	**1111111111000000**	**D/2**	16	**1111111111100100**
9/3	16	**1111111111000001**	**D/3**	16	**1111111111100101**
9/4	16	**1111111111000010**	**D/4**	16	**1111111111100110**
9/5	16	**1111111111000011**	**D/5**	16	**1111111111100111**
9/6	16	**1111111111000100**	**D/6**	16	**1111111111101000**
9/7	16	**1111111111000101**	**D/7**	16	**1111111111101001**
9/8	16	**1111111111000110**	**D/8**	16	**1111111111101010**
9/9	16	**1111111111000111**	**D/9**	16	**1111111111101011**
9/A	16	**1111111111001000**	**D/A**	16	**1111111111101100**
A/1	9	**111111000**	**E/1**	14	**11111111100000**
A/2	16	**1111111111001001**	**E/2**	16	**1111111111101101**
A/3	16	**1111111111001010**	**E/3**	16	**1111111111101110**
A/4	16	**1111111111001011**	**E/4**	16	**1111111111101111**
A/5	16	**1111111111001100**	**E/5**	16	**1111111111110000**
A/6	16	**1111111111001101**	**E/6**	16	**1111111111110001**
A/7	16	**1111111111001110**	**E/7**	16	**1111111111110010**
A/8	16	**1111111111001111**	**E/8**	16	**1111111111110011**
A/9	16	**1111111111010000**	**E/9**	16	**1111111111110100**
A/A	16	**1111111111010001**	**E/A**	16	**1111111111110101**
B/1	9	**111111001**	**F/1**	15	**111111111000011**
B/2	16	**1111111111010010**	**F/2**	16	**1111111111110110**
B/3	16	**1111111111010011**	**F/3**	16	**1111111111110111**
B/4	16	**1111111111010100**	**F/4**	16	**1111111111111000**
B/5	16	**1111111111010101**	**F/5**	16	**1111111111111001**
B/6	16	**1111111111010110**	**F/6**	16	**1111111111111010**
B/7	16	**1111111111010111**	**F/7**	16	**1111111111111011**
B/8	16	**1111111111011000**	**F/8**	16	**1111111111111100**
B/9	16	**1111111111011001**	**F/9**	16	**1111111111111101**
B/A	16	**1111111111011010**	**F/A**	16	**1111111111111110**
			F/0 (ZRL)	10	**1111111010**

Tabla A.6.- *Códigos base para la generación de coeficientes AC señal Crominancia.*

ANEXO B.- ***Arreglos implementados en el DSP para la realización de la etapa de Compresión***

Códigos_DC_Y [10] =

0	*2*	*3*	*4*	*5*	*6*	*14*	*30*	*62*	*126*

Tabla B.1*.- Arreglo de Códigos Huffman DC para Luminancia*

Longitud_DC_Y [10] =

2	*3*	*3*	*3*	*3*	*3*	*4*	*5*	*6*	*7*

Tabla B.2*.- Arreglo de longitudes DC para luminancia*

Códigos_AC_Y [16][10] =

0	*1*	*4*	*11*	*26*	*120*	*248*	*1014*	*65410*	*65411*
12	*27*	*121*	*502*	*2038*	*65412*	*65413*	*65414*	*65415*	*65416*
28	*249*	*1015*	*4084*	*65417*	*65418*	*65419*	*65420*	*65421*	*65422*
58	*503*	*4085*	*65423*	*65424*	*65425*	*65426*	*65427*	*65428*	*65429*
59	*1016*	*65430*	*65431*	*65432*	*65433*	*65434*	*65435*	*65436*	*65437*
122	*2039*	*65438*	*65439*	*65440*	*65441*	*65442*	*65443*	*65444*	*65445*
123	*4086*	*65446*	*65447*	*65448*	*65449*	*65450*	*65451*	*65452*	*65453*
250	*4087*	*65454*	*65455*	*65456*	*65457*	*65458*	*65459*	*65460*	*65461*
504	*32704*	*65462*	*65463*	*65464*	*65465*	*65466*	*65467*	*65468*	*65469*
505	*65470*	*65471*	*65472*	*65473*	*65474*	*65475*	*65476*	*65477*	*65478*
506	*65479*	*65480*	*65481*	*65482*	*65483*	*65484*	*65485*	*65486*	*65487*
1017	*65488*	*65489*	*65490*	*65491*	*65492*	*65493*	*65494*	*65495*	*65496*
1018	*65497*	*65498*	*65499*	*65500*	*65501*	*65502*	*65503*	*65504*	*65505*
2040	*65506*	*65507*	*65508*	*65509*	*65510*	*65511*	*65512*	*65513*	*65514*
65515	*65516*	*65517*	*65518*	*65519*	*65520*	*65521*	*65522*	*65523*	*65524*
65525	*65526*	*65527*	*65528*	*65529*	*65530*	*65531*	*65532*	*65533*	*65534*

Tabla B.3*.- Arreglo de Códigos Huffman AC para Luminancia*

Longitud_AC_Y [16][10] =

2	*2*	*3*	*4*	*5*	*7*	*8*	*10*	*16*	*16*
4	*5*	*7*	*9*	*11*	*16*	*16*	*16*	*16*	*16*
5	*8*	*10*	*12*	*16*	*16*	*16*	*16*	*16*	*16*
6	*9*	*12*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
6	*10*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
7	*11*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
7	*12*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
8	*12*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
9	*15*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
9	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
9	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
10	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
10	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
11	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
16	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
16	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*

Tabla B.4*.- Arreglo de longitudes AC para luminancia*

Códigos_DC_CrCb [10] =

0	*1*	*2*	*6*	*14*	*30*	*62*	*126*	*254*	*510*

***Tabla B.5**.- Arreglo de Códigos Huffman DC para Crominancia*

Longitud_DC_CrCb [10] =

2	*2*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*

***Tabla B.6**.- Arreglo de longitudes DC para Crominancia*

Códigos_AC_CrCb [16][10] =

1	*4*	*10*	*24*	*25*	*56*	*120*	*500*	*1014*	*4084*
11	*57*	*246*	*501*	*2038*	*4085*	*65416*	*65417*	*65418*	*65419*
26	*247*	*1015*	*4086*	*32706*	*65420*	*65421*	*65422*	*65423*	*65424*
26	*248*	*1016*	*4087*	*65425*	*65426*	*65427*	*65428*	*65429*	*65430*
58	*502*	*65431*	*65432*	*65433*	*65434*	*65435*	*65436*	*65437*	*65438*
59	*1017*	*65439*	*65440*	*65441*	*65442*	*65443*	*65444*	*65445*	*65446*
121	*2039*	*65447*	*65448*	*65449*	*65450*	*65451*	*65452*	*65453*	*65454*
122	*2040*	*65455*	*65456*	*65457*	*65458*	*65459*	*65460*	*65461*	*65462*
249	*65463*	*65464*	*65465*	*65466*	*65467*	*65468*	*65469*	*65470*	*65471*
503	*65472*	*65473*	*65474*	*65475*	*65476*	*65477*	*65478*	*65479*	*65480*
504	*65481*	*65482*	*65483*	*65484*	*65485*	*65486*	*65487*	*65488*	*65489*
505	*65490*	*65491*	*65492*	*65493*	*65494*	*65495*	*65496*	*65497*	*65498*
506	*65499*	*65500*	*65501*	*65502*	*65503*	*65504*	*65505*	*65506*	*65507*
2041	*65508*	*65509*	*65510*	*65511*	*65512*	*65513*	*65514*	*65515*	*65516*
16352	*65517*	*65518*	*65519*	*65520*	*65521*	*65522*	*65523*	*65524*	*65525*
32707	*65526*	*65527*	*65528*	*65529*	*65530*	*65531*	*65532*	*65533*	*65534*

***Tabla B.7**.- Arreglo de Códigos Huffman AC para Crominancia*

Longitud_AC_CrCb [16][10] =

2	*3*	*4*	*5*	*5*	*6*	*7*	*9*	*10*	*12*
4	*6*	*8*	*9*	*11*	*12*	*16*	*16*	*16*	*16*
5	*8*	*10*	*12*	*15*	*16*	*16*	*16*	*16*	*16*
5	*8*	*10*	*12*	*16*	*16*	*16*	*16*	*16*	*16*
6	*9*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
6	*10*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
7	*11*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
7	*11*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
8	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
9	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
9	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
9	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
9	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
11	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
14	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*
16	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*	*16*

***Tabla B.8**.- Arreglo de longitudes AC para Crominancia*

ANEXO C.- ***Arreglos implementados en el DSP para la realización de la etapa de Decompresión***

Longitud_DC_Y [7] =

2	*3*	*4*	*5*	*6*	*7*	*8*

Tabla C.1*.- Arreglo de longitudes DC para Luminancia*

SSSS_minimo_Y [7] =

0	*1*	*6*	*7*	*8*	*9*	*10*

Tabla C.2*.- Arreglo valores SSSS_mínimos DC para Luminancia*

Mínimo_DC_Y [7] =

0	*2*	*14*	*30*	*62*	*126*	*254*

Tabla C.3*.- Arreglo de valores mínimos DC para Luminancia*

Máximo_DC_Y [7] =

0	*6*	*14*	*30*	*62*	*126*	*254*

Tabla C.4*.- Arreglo de valores máximos DC para Luminancia*

Longitud_AC_Y [16][8] =

2	*3*	*4*	*5*	*7*	*8*	*10*	*16*
4	*5*	*7*	*9*	*11*	*16*	*17*	*17*
5	*8*	*10*	*12*	*16*	*17*	*17*	*17*
6	*9*	*12*	*16*	*17*	*17*	*17*	*17*
6	*10*	*16*	*17*	*17*	*17*	*17*	*17*
7	*11*	*16*	*17*	*17*	*17*	*17*	*17*
7	*12*	*16*	*17*	*17*	*17*	*17*	*17*
8	*12*	*16*	*17*	*17*	*17*	*17*	*17*
9	*15*	*16*	*17*	*17*	*17*	*17*	*17*
9	*16*	*17*	*17*	*17*	*17*	*17*	*17*
9	*16*	*17*	*17*	*17*	*17*	*17*	*17*
10	*16*	*17*	*17*	*17*	*17*	*17*	*17*
10	*16*	*17*	*17*	*17*	*17*	*17*	*17*
11	*16*	*17*	*17*	*17*	*17*	*17*	*17*
16	*17*	*17*	*17*	*17*	*17*	*17*	*17*
16	*17*	*17*	*17*	*17*	*17*	*17*	*17*

Tabla C.5*.- Arreglo de longitudes AC para Luminancia*

SSSS_minimo_AC_Y [16][8] =

1	*3*	*4*	*5*	*6*	*7*	*8*	*9*
1	*2*	*3*	*4*	*5*	*6*	*0*	*0*
1	*2*	*3*	*4*	*5*	*0*	*0*	*0*
1	*2*	*3*	*4*	*0*	*0*	*0*	*0*
1	*2*	*3*	*0*	*0*	*0*	*0*	*0*
1	*2*	*3*	*0*	*0*	*0*	*0*	*0*
1	*2*	*3*	*0*	*0*	*0*	*0*	*0*
1	*2*	*3*	*0*	*0*	*0*	*0*	*0*
1	*2*	*3*	*0*	*0*	*0*	*0*	*0*
1	*2*	*0*	*0*	*0*	*0*	*0*	*0*
1	*2*	*0*	*0*	*0*	*0*	*0*	*0*
1	*2*	*0*	*0*	*0*	*0*	*0*	*0*
1	*2*	*0*	*0*	*0*	*0*	*0*	*0*
1	*2*	*0*	*0*	*0*	*0*	*0*	*0*
1	*0*	*0*	*0*	*0*	*0*	*0*	*0*
1	*0*	*0*	*0*	*0*	*0*	*0*	*0*

***Tabla C.6**.- Arreglo de valores SSSS_mínimos AC para Luminancia*

Mínimo_AC_Y [16][8] =

0	*4*	*11*	*26*	*120*	*248*	*1014*	*65410*
12	*27*	*121*	*502*	*2038*	*65412*	*0*	*0*
28	*249*	*1015*	*4084*	*65417*	*0*	*0*	*0*
58	*503*	*4085*	*65423*	*0*	*0*	*0*	*0*
59	*1016*	*65430*	*0*	*0*	*0*	*0*	*0*
122	*2039*	*65438*	*0*	*0*	*0*	*0*	*0*
123	*4086*	*65446*	*0*	*0*	*0*	*0*	*0*
250	*4087*	*65454*	*0*	*0*	*0*	*0*	*0*
504	*32704*	*65462*	*0*	*0*	*0*	*0*	*0*
505	*65470*	*0*	*0*	*0*	*0*	*0*	*0*
506	*65479*	*0*	*0*	*0*	*0*	*0*	*0*
1017	*65488*	*0*	*0*	*0*	*0*	*0*	*0*
1018	*65497*	*0*	*0*	*0*	*0*	*0*	*0*
2040	*65506*	*0*	*0*	*0*	*0*	*0*	*0*
65515	*0*	*0*	*0*	*0*	*0*	*0*	*0*
65525	*0*	*0*	*0*	*0*	*0*	*0*	*0*

***Tabla C.7**.- Arreglo de valores mínimos AC para Luminancia*

Máximo_AC_Y [16][8] =

0	*4*	*11*	*26*	*120*	*248*	*1014*	*65411*
12	*27*	*121*	*502*	*2038*	*65416*	*0*	*0*
28	*249*	*1015*	*4084*	*65422*	*0*	*0*	*0*
58	*503*	*4085*	*65429*	*0*	*0*	*0*	*0*
59	*1016*	*65437*	*0*	*0*	*0*	*0*	*0*
122	*2039*	*65445*	*0*	*0*	*0*	*0*	*0*
123	*4086*	*65453*	*0*	*0*	*0*	*0*	*0*
250	*4087*	*65461*	*0*	*0*	*0*	*0*	*0*
504	*32704*	*65469*	*0*	*0*	*0*	*0*	*0*
505	*65478*	*0*	*0*	*0*	*0*	*0*	*0*
506	*65487*	*0*	*0*	*0*	*0*	*0*	*0*
1017	*65496*	*0*	*0*	*0*	*0*	*0*	*0*
1018	*65505*	*0*	*0*	*0*	*0*	*0*	*0*
2040	*65514*	*0*	*0*	*0*	*0*	*0*	*0*
65524	*0*	*0*	*0*	*0*	*0*	*0*	*0*
65534	*0*	*0*	*0*	*0*	*0*	*0*	*0*

***Tabla C.8**.- Arreglo de valores máximos AC para Luminancia*

Longitud_DC_CrCb [9] =

2	*3*	*4*	*5*	*6*	*7*	*8*	*9*

***Tabla C.9**.- Arreglo de longitudes DC para Crominancia*

SSSS_minimo_CrCb [9] =

0	*3*	*4*	*5*	*6*	*7*	*8*	*9*

***Tabla C.10**.- Arreglo de valores SSSS_mínimos DC para Crominancia*

Mínimo_DC_CrCb [9] =

0	*6*	*14*	*30*	*62*	*126*	*254*	*510*

***Tabla C.11**.- Arreglo de valores mínimos DC para Crominancia*

Máximo_DC_CrCb [9] =

2	*6*	*14*	*30*	*62*	*126*	*254*	*510*

***Tabla C.12**.- Arreglo de valores máximos DC para Crominancia*

Longitud_AC_CrCb [16][9] =

2	3	4	5	6	7	9	10
4	6	8	9	11	12	16	17
5	8	10	12	15	16	17	17
5	8	10	12	16	17	17	17
6	9	16	17	17	17	17	17
6	10	16	17	17	17	17	17
7	11	16	17	17	17	17	17
7	11	16	17	17	17	17	17
8	16	17	17	17	17	17	17
9	16	17	17	17	17	17	17
9	16	17	17	17	17	17	17
9	16	17	17	17	17	17	17
9	16	17	17	17	17	17	17
11	16	17	17	17	17	17	17
14	16	17	17	17	17	17	17
15	16	17	17	17	17	17	17

***Tabla C.13**.- Arreglo de longitudes AC para Crominancia*

SSSS_minimo_AC_CrCb [16][9] =

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

***Tabla C.14**.- Arreglo de valores SSSS_mínimos AC para Crominancia*

Mínimo_AC_CrCb [16][9] =

1	*4*	*10*	*24*	*56*	*120*	*500*	*1014*
11	*57*	*246*	*501*	*2038*	*4085*	*65416*	*0*
26	*247*	*1015*	*4086*	*32706*	*65420*	*0*	*0*
27	*248*	*1016*	*4087*	*65425*	*0*	*0*	*0*
58	*502*	*65431*	*0*	*0*	*0*	*0*	*0*
59	*1017*	*65439*	*0*	*0*	*0*	*0*	*0*
121	*2039*	*65447*	*0*	*0*	*0*	*0*	*0*
122	*2040*	*65455*	*0*	*0*	*0*	*0*	*0*
249	*65463*	*0*	*0*	*0*	*0*	*0*	*0*
503	*65472*	*0*	*0*	*0*	*0*	*0*	*0*
504	*65481*	*0*	*0*	*0*	*0*	*0*	*0*
505	*65490*	*0*	*0*	*0*	*0*	*0*	*0*
506	*65499*	*0*	*0*	*0*	*0*	*0*	*0*
2041	*65508*	*0*	*0*	*0*	*0*	*0*	*0*
16352	*65517*	*0*	*0*	*0*	*0*	*0*	*0*
32707	*65526*	*0*	*0*	*0*	*0*	*0*	*0*

***Tabla C.15**.- Arreglo de valores mínimos AC para Crominancia*

Máximo_AC_CrCb [16][9] =

1	*4*	*10*	*25*	*56*	*120*	*500*	*1014*
11	*57*	*246*	*501*	*2038*	*4085*	*65419*	*0*
26	*247*	*1015*	*4086*	*32706*	*65424*	*0*	*0*
27	*248*	*1016*	*4087*	*65430*	*0*	*0*	*0*
58	*502*	*65438*	*0*	*0*	*0*	*0*	*0*
59	*1017*	*65446*	*0*	*0*	*0*	*0*	*0*
121	*2039*	*65454*	*0*	*0*	*0*	*0*	*0*
122	*2040*	*65462*	*0*	*0*	*0*	*0*	*0*
249	*65471*	*0*	*0*	*0*	*0*	*0*	*0*
503	*65480*	*0*	*0*	*0*	*0*	*0*	*0*
504	*65489*	*0*	*0*	*0*	*0*	*0*	*0*
505	*65498*	*0*	*0*	*0*	*0*	*0*	*0*
506	*65507*	*0*	*0*	*0*	*0*	*0*	*0*
2041	*65516*	*0*	*0*	*0*	*0*	*0*	*0*
16352	*65525*	*0*	*0*	*0*	*0*	*0*	*0*
32707	*65534*	*0*	*0*	*0*	*0*	*0*	*0*

***Tabla C.16**.- Arreglo de valores máximos AC para Crominancia*

ANEXO D.- Programa utilizado por el Codificador MJPEG

```
#pragma DATA_SECTION(original_y,"mia");
#pragma DATA_SECTION(original_cb,"mia1");
#pragma DATA_SECTION(original_cr,"mia2");
#pragma DATA_SECTION(comprimida_ycrcb,"mia3");
#include<stdio.h>
#include<math.h>

//*****************************************//
        Declaración de los arreglos
    utilizados por la etapa de compresión
//*****************************************//

unsigned short var_conca=0,reg_temp=0,reg_duplicado;
unsigned char zrl,k,RRRR,SSSS,unidad_datos[8][8];
signed char
unidad_d_c[8][8],zig[64],yny=0,yncr=0,yncb=0,resul=16,longitud,dif;
short m=0,n=0,i,j;
unsigned int contador=0;
unsigned char
original_y[20][144][176],original_cb[20][144][176],original_cr[20][144][176
],cuadro;
unsigned short comprimida_ycrcb[506880];
float coeficientes[8][8],div=0.2;
//**************Declaración de subrutinas ******************************//
void Corrimiento(unsigned char unidad_datos[8][8], signed char
unidad_d_c[8][8]);
void DCT(signed char unidad_d_c[8][8], float coeficientes[8][8]);
void Cuantizacion_y(float coeficientes[8][8],signed char unidad_d_c[8][8]);
void Cuantizacion_crcb(float coeficientes[8][8],signed char
unidad_d_c[8][8]);
void Zigzag(signed char coeficientes[8][8],signed char zig[64]);
void Cod_Huffman_y();
void Cod_Huffman_crcb();
unsigned char categoria_dc(unsigned char);
unsigned char categoria_ac(unsigned char);
signed short bits_adicionales(signed short);
void concatenacion();

//***************************** main ************************//

void main(){
for(cuadro=0;cuadro<20;cuadro++){
longitud=l_code_SOI;
var_conca=code_SOI;
concatenacion();
yny=0;
yncr=0;
yncb=0;
n=0;
do{
m=0;
do{
for(j=0;j<8;j++){
for(i=0;i<8;i++)
unidad_datos[j][i]=
original_y[cuadro][j+n][i+m];}
Corrimiento(unidad_datos,unidad_d_c);
DCT(unidad_d_c,coeficientes);
Cuantizacion_y(coeficientes,unidad_d_c);
zig[0]=unidad_d_c[0][0]-yny;
Zigzag(unidad_d_c,zig);
Cod_Huffman_y();
for(j=0;j<8;j++){
for(i=0;i<8;i++)
unidad_datos[j][i]=
original_cr[cuadro][j+n][i+m];}
Corrimiento
     (unidad_datos,unidad_d_c);
DCT(unidad_d_c,coeficientes);
Cuantizacion_crcb
     (coeficientes,unidad_d_c);
zig[0]=unidad_d_c[0][0]-yncr;
yncr=unidad_d_c[0][0];
Zigzag(unidad_d_c,zig);
Cod_Huffman_crcb();
for(j=0;j<8;j++){
for(i=0;i<8;i++)
unidad_datos[j][i]=
original_cb[cuadro][j+n][i+m];}
```

```
yny=unidad_d_c[0][0];
Corrimiento(unidad_datos,unidad_d_c);
DCT(unidad_d_c,coeficientes);
Cuantizacion_crcb(coeficientes,
unidad_d_c);
zig[0]=unidad_d_c[0][0]-yncb;
yncb=unidad_d_c[0][0];
Zigzag(unidad_d_c,zig);
Cod_Huffman_crcb();

//*****************************//
m=m+8; } //primer while
while(m<176);
n=n+8; } //segundo while
while(n<144);
longitud=l_code_EOI;
var_conca=code_EOI;
concatenacion();} //
comprimida_ycrcb[contador]=reg_temp;
}//main


//*********************************//
void Cuantizacion_y(float x_coef[8][8],
signed char xc[8][8]){
short i,j;
for(j=0;j<=7;j++){
for(i=0;i<=7;i++)
xc[j][i]=(signed char)_roundf((div
*luminancia[j][i]*x_coef[j][i]));}}

void Cuantizacion_crcb
(float x_coef[8][8],signed char xc[8][8])
{
short i,j;
for(j=0;j<=7;j++){
for(i=0;i<=7;i++)
xc[j][i]=(signed char)_roundf((div
*crominancia[j][i]*x_coef[j][i]));}}

//*********************************//
void Cod_Huffman_y(){
zrl=0,k=0,RRRR=0;
SSSS=categoria_dc(zig[0]);
longitud=l_codes_DC_y[SSSS];
var_conca=codes_DC_y[SSSS];
concatenacion();
longitud=SSSS;
var_conca=bits_adicionales(zig[0]);
concatenacion();
incremento:k++;
if(zig[k]==0) { //if 1
if (k==63)
goto fin;
else { //else 2
RRRR=RRRR+1;
if(RRRR==16){
RRRR=0;
zrl=zrl+1;
goto incremento;}
else
goto incremento;}//else 2
}//if 1
else { //else 1
etiqueta_zrl: if(zrl>0){
longitud=l_code_ZRL_y;
var_conca=code_ZRL_y;
concatenacion();
zrl=zrl-1;
goto etiqueta_zrl;}
else {
SSSS=categoria_ac(zig[k]);
longitud=
l_codes_AC_y[RRRR][SSSS-1];
var_conca=
codes_AC_y[RRRR][SSSS-1];
concatenacion();
longitud=SSSS;
var_conca=
bits_adicionales(zig[k]);
concatenacion();
RRRR=0;}
if(k==63)
goto fin;
else
goto incremento;  }//else 1
fin: longitud=l_code_EOB_y;
var_conca=code_EOB_y;
concatenacion();  }

//*****************************//
void Cod_Huffman_crcb(){
zrl=0,k=0,RRRR=0;
SSSS=categoria_dc(zig[0]);
longitud=l_codes_DC_crcb[SSSS];
var_conca=codes_DC_crcb[SSSS];
concatenacion();
longitud=SSSS;
var_conca=
bits_adicionales(zig[0]);
concatenacion();
incremento:k++;
if(zig[k]==0) {
if (k==63)
goto fin;
else { //else 2
RRRR=RRRR+1;
if(RRRR==16){
RRRR=0;
zrl=zrl+1;
goto incremento;}
else
goto incremento;}//else 2 }//if 1
else { //else 1
etiqueta_zrl: if(zrl>0){
longitud=l_code_ZRL_crcb;
var_conca=code_ZRL_crcb;
concatenacion();
zrl=zrl-1;
goto etiqueta_zrl; }
else {
SSSS=categoria_ac(zig[k]);
longitud=
l_codes_AC_crcb[RRRR][SSSS-1];
```

```
var_conca=
codes_AC_crcb[RRRR][SSSS-1];
concatenacion();
longitud=SSSS;
var_conca=bits_adicionales(zig[k]);
concatenacion();
RRRR=0;}
if(k==63)
goto fin;
else
goto incremento; }//else 1
fin: longitud=l_code_EOB_crcb;
var_conca=code_EOB_crcb;
concatenacion();}//

//*****************************//
unsigned char
categoria_dc(unsigned char catego){
signed char comp=0;
if (zig[0]==0)
catego=0;
else{
if (zig[0]<0) {
comp=-zig[0];
catego=1;}
else {
comp=zig[0];
catego=1;}
while(comp>=
((unsigned char)powf(2,catego)))
catego=catego+1;}
return catego;}//

//*********************************//
unsigned char
categoria_ac(unsigned char catego){
signed char comp=0;
if (zig[k]<0) {
comp=-zig[k];
catego=1;}
else {
comp=zig[k];
catego=1;}
while(comp>=((unsigned char)
powf(2,catego)))
catego=catego+1;
return catego;}//

//*****************************//
signed short
bits_adicionales(signed short cantidad){
cantidad=(signed short)zig[k];
if (cantidad <0){
cantidad=-cantidad;
cantidad=~cantidad;
cantidad=cantidad<<(16-SSSS);
cantidad=cantidad>>(16-SSSS);}
else
cantidad=cantidad;
return cantidad;} //

//*****************************//
void concatenacion(){
resul=resul-longitud;
if(resul<=0)
if(resul==0){
reg_temp=reg_temp|var_conca;
comprimida_ycrcb[contador]=
reg_temp;
resul=16;
reg_temp=0;
contador=contador+1;}
else{
reg_duplicado=var_conca;
dif=-resul;
var_conca=var_conca>>dif;
reg_temp=reg_temp|var_conca;
comprimida_ycrcb[contador]=
reg_temp;
resul=16;
reg_temp=0;
contador=contador+1;
resul=resul-dif;
reg_duplicado=
reg_duplicado<<resul;
reg_temp=reg_temp|reg_duplicado;}
else {
var_conca=var_conca<<resul;
reg_temp=reg_temp|var_conca;}}//
//*****************************//
void DCT(signed char
unidad_desplazada[8][8],
float ydct[8][8]){
#define r   0.7071
#define r1  0.3536
float y[8], m_i_c[8][8];
short i,j;
void vector(float [], float []);
for(j=0;j<8;j++){
for(i=0; i<8; i++)
y[i]=(float)
unidad_desplazada[j][i];
vector(y,y);
for(i=0; i<8; i++)
m_i_c[j][i]=y[i];}
for(j=0;j<8;j++){
for(i=0; i<8; i++)
y[i]=m_i_c[i][j];
vector(y,y);
for(i=0; i<8; i++)
ydct[i][j]=y[i];}  } //
//*****************************//
```

```
void vector(float x[], float Xdct[] )
//sub funcion DCT unidimensional optimizada //{
float f,g,h,e,d,c,b,a;
h=x[0]+x[7];
g=x[1]+x[6];
f=x[2]+x[5];
e=x[3]+x[4];
d=x[0]-x[7];
c=x[1]-x[6];
b=x[2]-x[5];
a=x[3]-x[4];
Xdct[0]=.3536*(h+g+f+e);
Xdct[1]=0.4904*(d+(r*c)+(r*b))+0.0975*(a+(r*c)-(r*b));
Xdct[2]=0.4619*(h-e)+0.1913*(g-f);
Xdct[3]=0.2778*((r*c)-(r*b)-a)+0.4157*(d-(r*c)-(r*b));
Xdct[4]=r1*(h+e-g-f);
Xdct[5]=0.2778*(d-(r*c)-(r*b))+0.4157*(a-(r*c)+(r*b));
Xdct[6]=0.1913*(h-e)+0.4619*(f-g);
Xdct[7]=0.0975*((r*c)+(r*b)+d)+0.4904*((r*b)-(r*c)-a);
} /// DCT unidimensional optimizada //

//*****************************************************//
void Zigzag(signed char unidad_datos[8][8],signed char arreglo[64]){
short j=0,i=0,k=0,n,indice=1;

for(indice=1;(indice<7);indice=indice+2){
  arreglo[k=k+1]=unidad_datos[j][i=i+1];

for(n=0;(n<indice);n++)
  arreglo[k=k+1]=unidad_datos[j=j+1][i=i-1];
  arreglo[k=k+1]=unidad_datos[j=j+1][i];

for(n=0;((n<indice+1));n++)
  arreglo[k=k+1]=unidad_datos[j=j-1][i=i+1];  }
  arreglo[k=k+1]=unidad_datos[j][i=i+1];

for(n=0;(n<7);n++)
  arreglo[k=k+1]=unidad_datos[j=j+1][i=i-1];

for(indice=5;(indice>=1);indice=indice-2){
  arreglo[k=k+1]=unidad_datos[j][i=i+1];

for(n=0; ((n<indice+1));n++)
  arreglo[k=k+1]=unidad_datos[j=j-1][i=i+1];
  arreglo[k=k+1]=unidad_datos[j=j+1][i];

for(n=0; (n<indice);n++)
  arreglo[k=k+1]=unidad_datos[j=j+1][i=i-1];  }
  arreglo[k=k+1]=unidad_datos[j][i=i+1];

//*****************************************************//
void Corrimiento(unsigned char x[8][8], signed char x1[8][8])
{signed char i,j;
for(i=0;i<=7;i++){
for(j=0;j<=7;j++)
x1[i][j]=x[i][j]-128; }}
```

ANEXO E.-

Programa utilizado por el Decodificador MJPEG

```
#pragma DATA_SECTION(original_y,"mia");
#pragma DATA_SECTION(original_cb,"mia1");
#pragma DATA_SECTION(original_cr,"mia2");
#pragma DATA_SECTION(comprimida_ycrcb,"mia3");
#include<stdio.h>
#include<math.h>
//*****************************************************************//
       Declaración de los arreglos utilizados por la Decompresión
//*****************************************************************/
void lectura();
void codigo_huffman_DC();
void codigo_huffman_AC();
void bits_adicionales();
void decodificar_DPCM();
void decodificar_AC();
void codigo_huffman_DC_crcb();
void codigo_huffman_AC_crcb();
void decodificar_DPCM_crcb();
void decodificar_AC_crcb();
void Dzigzag(char zig[64], signed char unidad_datos[8][8]);
void D_cuantizacion();
void D_cuantizacion_crcb();
void IDCT(float [8][8], float [8][8]);
void Corrimiento();

char bits_dec=0,bits_leidos=2,acum;
unsigned short reg1=0,reg2=0,reg3=0,contador=0,var,comparacion;
unsigned char dif=0,SSSS=0,var1=0,unidad[8][8];
char cantidad=0,eob=0,zrl=0,RRRR=0,k=0,zig[64],yn=0,yn_cr=0,yn_cb=0;
char eob_crcb=0,zrl_crcb=0;
short m=0,n=0,i,j;
unsigned char
original_y[20][144][176],original_cb[20][144][176],original_cr[20][144][176
],cuadro;
unsigned short comprimida_ycrcb[506880];
 signed char unidad_datos[8][8];
float coeficientes[8][8],div=0.2;



void main(){

for(cuadro=0;cuadro<20;cuadro++){

yn=0;
yn_cr=0;
yn_cb=0;
n=0;
do{
m=0;
do {
decodificar_DPCM();
decodificar_AC();
zig[0]=zig[0]+yn;
yn=zig[0];
Dzigzag(zig,unidad_datos);
D_cuantizacion();
IDCT(coeficientes,coeficientes);
Corrimiento();
for(j=0;j<8;j++){
for(i=0;i<8;i++)
original_y[cuadro][j+n][i+m]
=unidad[j][i];}
decodificar_DPCM_crcb();
```

```
decodificar_AC_crcb();
zig[0]=zig[0]+yn_cr;
yn_cr=zig[0];
Dzigzag(zig,unidad_datos);1
D_cuantizacion_crcb();
IDCT(coeficientes,coeficientes);
Corrimiento();
for(j=0;j<8;j++){
for(i=0;i<8;i++)
original_cr[cuadro][j+n][i+m]=
unidad[j][i];}
decodificar_DPCM_crcb();
decodificar_AC_crcb();
zig[0]=zig[0]+yn_cb;
yn_cb=zig[0];
Dzigzag(zig,unidad_datos);
D_cuantizacion_crcb();
IDCT(coeficientes,coeficientes);
Corrimiento();
for(j=0;j<8;j++){
for(i=0;i<8;i++)
original_cb[cuadro][j+n][i+m]=
unidad[j][i];}
m=m+8; } //primer while
while(m<176);

n=n+8;  //segundo while
while(n<144);}}

//************* main **************//
void D_cuantizacion() {
float factor;
short i,j;
factor=1/div;
for(j=0;j<=7;j++){
for(i=0;i<=7;i++){
coeficientes[j][i]=
(float)
luminancia[j][i]*unidad_datos[j][i];
coeficientes[j][i]=
factor*coeficientes[j][i];}}}

//*********************************//
void D_cuantizacion_crcb() {
float factor;
short i,j;
factor=1/div;
for(j=0;j<=7;j++){
for(i=0;i<=7;i++){
coeficientes[j][i]=
(float)
crominancia[j][i]*unidad_datos[j][i];
coeficientes[j][i]=
factor*coeficientes[j][i];}}}

//*********************************//
void Corrimiento(){
signed char i,j;
for(i=0;i<=7;i++){
for(j=0;j<=7;j++)
unidad[j][i]=
(unsigned char)
  coeficientes[j][i]+128; }}

//*****************************//
void lectura(){//*
bits_dec=bits_dec+bits_leidos;
if(bits_dec <= 16){
reg1=comprimida_ycrcb[contador];
reg1=reg1 >>(16-bits_dec);
var=powf(2,bits_leidos)-1;
reg2=reg2 |(reg1 & var);}
if(bits_dec>16){
bits_dec=bits_dec-16;
reg1=comprimida_ycrcb[contador];
reg1=reg1 << (bits_dec);
var=powf(2,bits_leidos)-1;
reg1= reg1 & var;
contador=contador+1;
reg3=comprimida_ycrcb[contador];
reg3=reg3>>(16-bits_dec);
reg2=reg1|reg3; } }//*


//***************************//
void codigo_huffman_DC(){//*
unsigned char i=0;
reg2=0;
bits_leidos=2;
acum=2;
algoritmo_lectura:
lectura();
var=powf(2,acum)-1;
if (var==reg2){
regresar:
if(bits_dec>1){
bits_dec=bits_dec-bits_leidos;
bits_leidos=bits_leidos+1;
reg2=reg2<<1;
acum=acum+1;
goto algoritmo_lectura;}
else {
bits_leidos=1;
reg2=reg2<<1;
acum=acum+1;
goto algoritmo_lectura;}}
else {
pruebas:
if(longitud_DC_Y[i]>acum)
goto regresar;
if(longitud_DC_Y[i]<acum)
i=i+1;
if(longitud_DC_Y[i]==acum) {
if(maximo_DC_Y[i]>=reg2)
goto fin;
else {
i=i+1;
goto pruebas;} }
```

```
else
goto pruebas;}
fin:
dif=(unsigned char)reg2-minimo_DC_Y[i];
SSSS=dif+SSSS_minimo[i];}//*

//*****************************//
void codigo_huffman_AC(){//*
char i=0,j=0;
reg2=0;
bits_leidos=2;
acum=2;
saltar:
lectura();
var=powf(2,acum)-1;
if (var==reg2){ //if1
back: if(bits_dec >1){
bits_dec=bits_dec-bits_leidos;
bits_leidos=bits_leidos+1;
reg2=reg2 <<1;
acum=acum+1;
goto saltar;}
else{
bits_leidos=1;
reg2=reg2 << 1;
acum=acum+1;
goto saltar;}}//if1
else {//else1
if(reg2==10) {
eob=1;
goto nextstage; }
if(reg2==2041){
zrl=1;
goto nextstage; }
otravez: if(longitud_AC_Y[i][j]>acum) {
i=i+1;
j=0;
if(i==16){
i=0;
goto back;}}
if(longitud_AC_Y[i][j]<acum)
j=j+1;
if(longitud_AC_Y[i][j]==acum){
if(maximo_AC_Y[i][j]>=reg2)
goto finish;
else {
j=j+1;
if(j==8){
i=i+1;
j=0;
goto otravez;}
goto otravez;}}
else
goto otravez;}//else1
finish:
dif=
(unsigned char)reg2-minimo_AC_Y[i][j];
SSSS=dif+SSSS_min_AC_Y[i][j];
RRRR=i;
nextstage:}//*

//*****************************//
void bits_adicionales(){//*
bits_leidos=SSSS;
reg2=0;
lectura();
comparacion=reg2;
cantidad=(char)reg2;
comparacion=comparacion>>(SSSS-1);
if(comparacion==1)
cantidad=cantidad;
else {
var1=powf(2,SSSS)-1;
cantidad=~cantidad;
cantidad=cantidad & var1;
cantidad=-cantidad; }} //*

//*****************************//
void decodificar_DPCM(){//*
codigo_huffman_DC();
bits_adicionales();
zig[k]=cantidad; }//*

//*****************************//
void decodificar_AC(){//*
nexttime: codigo_huffman_AC();
if(eob==1) { //if1
RRRR=63-k;
aver: if(RRRR==0)
goto acabar;
else{
k=k+1;
zig[k]=0;
RRRR=RRRR-1;
goto aver;} } //if1
else  { //else1
if(zrl==1) { //if2
RRRR=16;
again: if(RRRR==0)
{ zrl=0;
goto nexttime;}
else{
k=k+1;
zig[k]=0;
RRRR=RRRR-1;
goto again;} }//if2
else {//else2
bits_adicionales();
yamecanse: if(RRRR==0){
k=k+1;
zig[k]=cantidad;
goto nexttime;}
else {
k=k+1;
zig[k]=0;
RRRR=RRRR-1;
```

```
goto yamecanse;} }//else2} //else1
acabar:
k=0;
eob=0;}//*

//*********************************//
void codigo_huffman_DC_crcb(){//**
unsigned char i=0;
reg2=0;
bits_leidos=2;
acum=2;
algoritmo_lectura:
lectura();
var=powf(2,acum)-1;
if (var==reg2)  {
regresar:
if(bits_dec>1){
bits_dec=bits_dec-bits_leidos;
bits_leidos=bits_leidos+1;
reg2=reg2<<1;
acum=acum+1;
goto algoritmo_lectura;}
else {
bits_leidos=1;
reg2=reg2<<1;
acum=acum+1;
goto algoritmo_lectura;} }
else {
pruebas:
if(longitud_DC_crcb[i]>acum)
goto regresar;
if(longitud_DC_crcb[i]<acum)
i=i+1;
if(longitud_DC_crcb[i]==acum) {
if(maximo_DC_crcb[i]>=reg2)
goto fin;
else {
i=i+1;
goto pruebas;}}
else
goto pruebas;}
fin:
dif=
(unsigned char)reg2-minimo_DC_crcb[i];
SSSS=dif+SSSS_minimo_crcb[i];}//**

//*********************************//
void codigo_huffman_AC_crcb(){//**
char i=0,j=0;
reg2=0;
bits_leidos=2;
acum=2;
saltar:
lectura();
var=powf(2,acum)-1;
if (var==reg2) { //if1
back: if(bits_dec>1){
bits_dec=bits_dec-bits_leidos;
bits_leidos=bits_leidos+1;
reg2=reg2<<1;
acum=acum+1;
goto saltar;}
else{
bits_leidos=1;
reg2=reg2 << 1;
acum=acum+1;
goto saltar;} }//if1
else {//else1
if(reg2==0){
eob_crcb=1;
goto nextstage;}
if(reg2==1018){
zrl_crcb=1;
goto nextstage;}
otravez:
if(longitud_AC_crcb[i][j]>acum)
{ i=i+1;
j=0;
if(i==16){
i=0;
goto back;}}
if(longitud_AC_crcb[i][j]<acum)
j=j+1;
if(longitud_AC_crcb[i][j]==acum){
if(maximo_AC_crcb[i][j]>=reg2)
goto finish;
else
{ j=j+1;
if(j==9){
i=i+1;
j=0;
goto otravez;}
goto otravez;  }}
else
goto otravez; }//else1
finish:
dif=(unsigned char)reg2 -
minimo_AC_crcb[i][j];
SSSS=dif+SSSS_min_AC_crcb[i][j];
RRRR=i;
nextstage:}//**

//*****************************//
void decodificar_DPCM_crcb(){//**
codigo_huffman_DC_crcb();
bits_adicionales();
zig[k]=cantidad;} //**

//*****************************//
void decodificar_AC_crcb() {//**
nexttime:codigo_huffman_AC_crcb();
if(eob_crcb==1) { //if1
RRRR=63-k;
aver: if(RRRR==0)
goto acabar;
else{
```

```
k=k+1;
zig[k]=0;
RRRR=RRRR-1;
goto aver;}} //if1
else{ //else1
if(zrl_crcb==1) { //if2
RRRR=16;
again: if(RRRR==0){
zrl_crcb=0;
goto nexttime;}
else{
k=k+1;
zig[k]=0;
RRRR=RRRR-1;
goto again;}} //if2
else {//else2
bits_adicionales();
yamecanse: if(RRRR==0){
k=k+1;
zig[k]=cantidad;
goto nexttime;}
else {
k=k+1;
zig[k]=0;
RRRR=RRRR-1;
goto yamecanse;}}}
acabar: k=0;eob_crcb=0;}//**

//*********************************//
void IDCT(float unidad_d[8][8],
float yidct[8][8]){
#define r   0.7071
#define r1  0.3536
float y[8], m_i_c[8][8];
short i,j;
void vector(float [], float[]);
for(j=0;j<8;j++){
for(i=0; i<8; i++)
y[i]= unidad_d[j][i];
vector(y,y);
for(i=0; i<8; i++)
m_i_c[j][i]=y[i];}
for(j=0;j<8;j++){
for(i=0; i<8; i++)
y[i]=m_i_c[i][j];
vector(y,y);
//*******************************//
for(i=0; i<8; i++)
yidct[i][j]=_roundf(y[i]);}}
void vector(float x[], float Idct[]){
float a,b,c,d,e,f,g,h;
a=.4904*(x[1]+(r*x[3])+(r*x[5]))+.0975*(x[7]+(r*x[3])-(r*x[5]));
b=.4157*(x[1]-(r*x[3])-(r*x[5]))-.2778*(x[7]-(r*x[3])+(r*x[5]));
c=.2778*(x[1]-(r*x[3])-(r*x[5]))+.4157*(x[7]-(r*x[3])+(r*x[5]));
d=.0975*(x[1]+(r*x[3])+(r*x[5]))-.4904*(x[7]+(r*x[3])-(r*x[5]));
e=(.4619*x[2])+(.1913*x[6])+(r1*x[0])+(r1*x[4]);
f=(.1913*x[2])-(.4619*x[6])+(r1*x[0])-(r1*x[4]);
g=-(.1913*x[2])+(.4619*x[6])+(r1*x[0])-(r1*x[4]);
h=-(.4619*x[2])-(.1913*x[6])+(r1*x[0])+(r1*x[4]);
Idct[0]=a+e;
Idct[1]=b+f;
Idct[2]=c+g;
Idct[3]=d+h;
Idct[4]=h-d;
Idct[5]=g-c;
Idct[6]=f-b;
Idct[7]=e-a;}

//*******************************//
void Dzigzag(char arreglo[64],
signed char unidad_datos[8][8]){
short j=0,i=0,k=0,n,indice=1;
unidad_datos[j][i]=arreglo[k];
for(indice=1;(indice<7);
indice=indice+2){
unidad_datos[j][i=i+1]=
arreglo[k=k+1];
for(n=0;(n<indice);n++)
unidad_datos[j=j+1][i=i-1]=
arreglo[k=k+1];
unidad_datos[j=j+1][i]=
arreglo[k=k+1];
for(n=0;((n<indice+1));n++)
unidad_datos[j=j-1][i=i+1]=
arreglo[k=k+1]; }
unidad_datos[j][i=i+1]=
arreglo[k=k+1];
for(n=0;(n<7);n++)
unidad_datos[j=j+1][i=i-1]=
arreglo[k=k+1];
for(indice=5;(indice>=1
);indice=indice-2) {
unidad_datos[j][i=i+1]=
arreglo[k=k+1];
for(n=0; ((n<indice+1));n++)
unidad_datos[j=j-1][i=i+1]=
arreglo[k=k+1];
unidad_datos[j=j+1][i]=
arreglo[k=k+1];
for(n=0; (n<indice);n++)
unidad_datos[j=j+1][i=i-1]=
arreglo[k=k+1]; }
unidad_datos[j][i=i+1]=
arreglo[k=k+1];
```

BIBLIOGRAFÍA

[1] **JPEG STILL DATA COMPRESIÓN STANDARD**, *William B. Pennebaker, Joan L. Mitchell, Chapman & Hall, Digital Multimedia Standards, NuevaYork, N.Y, 1993*

[2] **Video Coding for Mobile Communications, Efficiency Complexity and Resillence** , *Mohammed Ebrahim Al – Mualla, C. Nishan Canagarajah and David R.Bull, Academic Press, San Diego Cal, 2001*

[3] **Compressed Video Over Networks**, *Ming –Ting Sun Amy R. Reibman, Marcel Dekker, Nueva York, N.Y, 2001*

[4] **Discrete cosine transform algorithms, advantages, applications,** *K.R.Rao, P.Yip, Academic Press, San Diego CA, 1990.*

[5] **Fast transforms algorithms, advantage, applications**, *Douglas F. Elliot, K Ramamohan Rao, Academic Press, Orlando Florida, 1982.*

[6] **Compresión de imágenes utilizando la transformada Coseno en un DSP**, *Riwes Cruz Jesús Hernández, Tesis México D.F, 2004.*

[7] **Teoría del color**
http: // www.juanval.net/colordigital.htm

[8] **Visión por computador (imágenes digitales y aplicaciones),** *Pajares Gonzalo, De la Cruz Jesús, Grupo Editorial Alfaomega, México D.F, 2004*

[9] **Manual de fotografía,** *Tomas Ang, Editorial Georgiona Garner, ediciones Omega, Londres 2002*

[10] **Data compression in digital systems**, *Roy Hoffman, Chapman & Hall Nueva York, N.y, 1996*

[11] **The application of programmable DSP's in mobile communications,** *Alan Gatherer and Edgar Auslander, Chichester Inglaterra, 2002*

[13] **The transform and data compression handbook,** *K.R.Rao and P.C. Yip, The Electrical Enginnering and Signal Processing series, CRC PRESS Florida, E.U, 2001*

[14] **The JPEG STILL PICTURES COMPRESSION STANDARD**.*Gregory K. Wallace, IEEE Transactions on consumer electronics Vol 38, no 1, February 1992.*

[15] **Televisión práctica y sistemas de video**, *Grob, Herdon Grupo editorial Alfaomega 6ª Edición, México D.F, 1992*

[16] **Standar Codecs : image compression to advanced video coding**, *Mohammed Ghanbari ,IEEE Telecommunications serie 49, Londres, Inglaterra, 2002*

[17] **Diccionario de sinónimos y antónimos,** *Editorial Océano. México, D.F, 1994*

[18] **The MPEG handbook,** *John Watkinson, Second Edition, Focal Press Oxford, Inglaterra, 2004.*

[19] **Digital video: An introducction to MPEG-2,** *Barry G. Haskell, Atul Puri, and Arun N. Netravali, Digital Multimedia Standards Series Florida, E.U, 1997.*

[20] **La tesis: Manual para la elaboración de tesis,** *Francisco Beverido P, Textos universitarios, Universidad Veracruzana, Xalapa, Veracruz, Abril 1993*

[21] **TMS320C6000 CPU and Instruction Set,** *Reference Guide Digital Signal Processing Solutions, Texas Instruments, October 2000*

[22] **TMS320C64x,** *Technical Overview Digital Signal Processing Solutions, Texas Instruments, February 2000*

[23] **Como programar en C/C++,** *H.M DEITEL/P,J DEITEL Prentice Hal, segunda edición, México D.F, 1997*

[24] **WWW.TI.COM**

GLOSARIO

Algoritmo: Secuencia finita de operaciones realizables, cuya ejecución da una solución de un problema en un tiempo finito.

Aliasing: Efecto indeseable que causa que señales continuas se tornen indistinguibles cuando se les muestrea digitalmente, cuando esto sucede la señal original no puede ser reconstruida de forma unívoca a partir de la señal digital.

Ancho de banda: Intervalo de frecuencias en el que se concentra la mayor parte de la potencia de una señal.

ATM: Modo de Transferencia Asíncrona (ATM) por sus siglas en inglés, es una tecnología de telecomunicaciones desarrollada para hacer frente a la gran demanda de capacidad de transmisión para servicios y aplicaciones. En este sistema la información no se transmite y se conmuta a través de canales asignados en permanencia, sino en forma de paquetes cortos (celdas ATM) de longitud constante.

Audio: Representación eléctrica de una señal sonora, normalmente está acotada al intervalo de frecuencias audibles por los seres humanos que están entre los 20 y los 20,000 Hertzs, aproximadamente.

Bit: Acrónimo de digito binario, es un dígito del sistema de numeración binario al cual se le puede asignar solo dos valores 0 y 1. El bit es la unidad mínima de información empleada en informática, y en cualquier dispositivo digital.

Camcorder: Proviene de las palabras *camera–recorder* con la cual se designa a dispositivos que realizan la captura y la grabación de la señal de video en un mismo equipo.

CCITT: Son las siglas de Comité Consultivo Internacional Telegráfico y Telefónico, antiguo nombre del comité de normalización de las telecomunicaciones.

CD: Acrónimo de disco compacto, es un soporte digital óptico utilizado para almacenar cualquier tipo de información (audio, video, documentos). Fue desarrollado conjuntamente en 1980 por las empresas Sony y Philips, y comenzó a comercializarse en 1982.

Crominancia*:* Componente de la señal de vídeo que contiene la información del color.

DCT: Transformada Coseno Discreta es una transformación matemática basada, en la Transformada de Fourier discreta, pero utiliza únicamente números reales.

DFT: Acrónimo de Transformada Discreta de Fourier, proceso que nos permite encontrar las frecuencias que componen a una señal discreta.

DSP: Acrónimo de *Procesador Digital de Señal* por sus siglas en Inglés, es un sistema basado en un procesador o microprocesador que posee un conjunto de instrucciones y hardware optimizados para aplicaciones que requieren operaciones numéricas a muy alta velocidad. Es especialmente útil para el procesado y representación de señales analógicas en tiempo real.

H.261: Codificador estándar de video originalmente diseñado para transmisión sobre redes ISDN, a razones de bit que fueran múltiplos de 64 Kbit/s, fue el primer codificador estándar de video desarrollado.

IDCT: Acrónimo de Transformada discreta Coseno Inversa, por sus siglas en Inglés, se refiere al proceso mediante el cual se recuperan las muestras a partir de un conjunto de coeficientes o amplitudes de las funciones base.

ISDN: Red Digital de Servicios Integrados por sus siglas en Inglés: una red que facilita conexiones digitales extremo a extremo para proporcionar una amplia gama de servicios, tanto de voz como de otros tipos, y a la que los usuarios acceden a través de un conjunto de interfaces normalizados.

JPEG: Acrónimo de Unión Grupal de Expertos en fotografía, este grupo creó un algoritmo de compresión de imágenes en tono continuo al cual designaron con el mismo nombre.

KLT: Acrónimo de Transforma Karhunen Loeve, transformación que decorrelaciona una función aleatoria y entrega coeficientes dentro del dominio de la transformada.

Luminancia: Es la componente que codifica la información de luminosidad de la imagen.

MJPEG: Acrónimo de JPEG en movimiento, es un codificador de video digital donde cada cuadro de la secuencia es codificado separadamente utilizando el algoritmo de compresión JPEG.

MPEG1: Grupo de Expertos en imágenes en Movimiento, el cual da el mismo nombre al algoritmo de compresión de video que se utiliza en el formato Video CD. La calidad de salida junto a su tasa de compresión usual usada en el video CD es similar a la de un cassette video VHS doméstico.

MPEG2: Uno de los formatos de compresión más sofisticados, MPEG-2 es típicamente usado para codificar y transmitir audio-video por Satélite Digital y Cable. MPEG-2, con algunas modificaciones es utilizado como él formato de codificación usado por el DVD para películas comerciales. Es similar a MPEG1, pero también proporciona soporte para video entrelazado (el formato utilizado por las televisiones).

MPEG4: Introducido a finales de 1998, es el nombre de un grupo de estándares de codificación de audio y video, los usos principales del estándar MPEG4 son los medios audiovisuales, la distribución en CD, la transmisión bi-direccional por videoteléfono y el intercambio entre distintos tipos de redes digitales.

NTSC: Sistema de codificación y transmisión de televisión analógica desarrollado en Estados Unidos alrededor de 1940, y que se emplea en la actualidad en la mayor parte de América, Japón y algunos

otros países. El nombre viene del comité de expertos que lo desarrolló, el Comité Nacional de Sistemas de Televisión por sus siglas en Inglés.

PAL: Acrónimo de línea alternada en fase, es el nombre con el que se designa al sistema de codificación empleado en la transmisión de señales de televisión analógica en color en la mayor parte del mundo. Es de origen alemán y se utiliza en la mayoría de los países africanos, asiáticos, europeos, además de Australia y algunos países latinoamericanos.

PDS**:** Abreviatura de Procesamiento Digital de Señales, área de la ingeniería que se dedica al análisis y procesamiento de señales discretas.

Píxel: Elemento de imagen, es la menor unidad en la que se descompone una imagen digital, ya sea una fotografía, un cuadro de video o un gráfico.

PSTN: Acrónimo de Red Pública Telefónica Conmutada por sus siglas en Inglés, es una red de teléfono diseñada primordialmente para la transmisión de voz, aunque también puede transportar datos, por ejemplo el caso del fax o de la conexión a Internet a través de un módem.

RGB: modelo aditivo de color, en el cual los colores rojo, verde, y azul son combinados de diferentes maneras para crear otros colores.

SDTV: Acrónimo de Televisión Estándar Digital por sus siglas en Inglés, sistema desarrollado para la transmisión digital de video, algunas normas de este sistema pueden tener la misma resolución que los sistemas analógicos, NTSC, PAL, SECAM.

SECAM : Acrónimo de Color secuencial con memoria por sus siglas en Francés, es un sistema para la codificación de televisión en color analógica utilizado por primera vez en Francia, es históricamente la primera norma de televisión en color Europea.

YUV: Espacio de color en términos de una componente de luminancia y dos componentes de crominancia, utilizado en los sistemas PAL y NTSC de difusión de televisión.