UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

FACULTAD DE INGENIERÍA

IMPLEMENTACIÓN DE UN SISTEMA DE ADQUISICIÓN Y COMPRESIÓN DE VIDEO MJPEG EN UN DSP

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN INGENIERÍA

INGENIERÍA ELÉCTRICA - SISTEMAS ELECTRÓNICOS

P R E S E N T A :

EDGAR ALONSO TRUEBA

TUTOR:

M.I. LARRY HIPÓLITO ESCOBAR SALGUERO

2009

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

JURADO ASIGNADO:

Presidente:	Dr. en Ing. Pérez Alcázar Pablo Roberto
Secretario:	M. I. Quintana Thierry Sergio
Vocal:	M. I. Escobar Salguero Larry H.
1er. Suplente:	DR. Psenicka Bohumil
2do. Suplente:	M. I. Damián Zamacona Ricardo

Lugar o lugares donde se realizó la tesis:

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

TUTOR DE TESIS:

M. I. Escobar Salguero Larry H.

_____

FIRMA

Agradecimientos:

A mis padres, que todo el tiempo están conmigo, porque las personas que amas nunca mueren.

A toda mi familia, por todo el apoyo brindado para que pudiera alcanzar esta meta, y con la que siempre estaré en deuda.

A la Universidad Nacional Autónoma de México, por permitirme formar parte de ella y brindarme todo lo necesario durante mis estudios.

Al M. I. Larry Escobar Salguero, por todo su valioso apoyo y amistad que me brindo para poder concluir la Maestría y tener una formación académica integra.

A mis compañeros y profesores de Maestría, quienes se han esforzado porque la opción de Maestría de Sistemas Electrónicos siga vigente.

A todos mis compañeros de laboratorio, en especial al Ing. Pedro Jiménez por su colaboración en el desarrollo del reproductor de video de este proyecto.

A mis sinodales, por sus observaciones y recomendaciones para mejorar la calidad de este trabajo.

A todos mis amigos, y a todas aquellas personas que de alguna forma contribuyeron para alcanzar este logro, muchas gracias.

ÍNDICE

5.- IMPLEMENTACION DEL COMPRESOR DE VIDEO MJPEG EN EL DSP C6416

6.- EVALUACION DE LOS RESULTADOS OBTENIDOS

INTRODUCCION

Los sistemas multimedia actuales forman parte de la vida cotidiana y los podemos encontrar en aplicaciones como video conferencias, cámaras fotográficas y de video, cámaras web, servidores de video que usan el *Protocolo de Internet (IP)*, sistemas de vigilancia, entre otros. Estos sistemas procesan una gran cantidad de información que es generada por las imágenes y el video, y requieren un gran ancho de banda, por lo que se tiene la necesidad de comprimir la información para transmitirla o almacenarla, tratando de conservar la calidad perceptible por el ojo humano.

En un sistema de transmisión o almacenamiento de video digital, se requiere de un proceso de adquisición de imágenes, así como uno de compresión de datos. Este tipo de sistemas por lo regular operan en tiempo real y se necesita de la integración de un sensor de video con un hardware apropiado de alto desempeño, aunado a un software eficiente que sincronice, controle y realice adecuadamente los procesos de adquisición, compresión y transmisión de video.

El tipo y el porcentaje de compresión dependen de la aplicación, por ejemplo en sistemas de transmisión se suele utilizar una compresión muy alta, ya que el ancho de banda de los canales es reducido. En los ambientes de producción y almacenamiento de películas en *DVD (Digital Versatile Disc)* se utiliza menos compresión con el fin de mantener una buena calidad de imagen en todas las fases hasta tener el master final editado. Todos los métodos de compresión están basados en el principio de eliminación de la información que es menos perceptible por el ojo humano; los denominados detalles "redundantes" de la imagen. Esta técnica se aplica tanto a imágenes estáticas como al material de vídeo y cine, utilizándose en la actualidad varias técnicas combinadas [1], [8].

La percepción de color (crominancia) del ojo humano, no es tan precisa como la del blanco y negro (luminancia), así que la resolución del color se puede reducir en un 50% con respecto a la luminancia (formato de submuestreo 4:2:2) o en un 75% (formato de submuestreo 4:2:0). Esta técnica se utiliza en algunos sistemas de televisión a color. De modo similar, los detalles de tamaño pequeño con poco contraste resultan menos perceptibles que los objetos grandes con mucho contraste [1].

La compresión de imágenes de video en el dominio espacial disminuye la redundancia entre pixeles de una imagen y se conoce como *codificación espacial* o *Intraframe*. En la mayoría de los estándares de compresión de imágenes y video que utilizan codificación espacial se utiliza un proceso denominado *Transformada Coseno Discreta (DCT por sus siglas en inglés)*; en la mayoría de los casos la imagen se divide en bloques de 8 x 8 pixeles, para que sea posible reducir la escala o cuantizar los coeficientes DCT y así reducir la cantidad de datos. Esta técnica se utiliza en muchos esquemas de compresión de imágenes y video digital actuales, como *Video Digital (DV), Video de Alta Definición (HDV), JPEG (Join Photograph Expert Group), I frames de MPEG-1 y MPEG-2 (MPEG - Motion Pictures Experts Group)* y *Windows Media*. También se realiza una reducción adicional mediante la codificación entrópica, un proceso matemático que descarta la información redundante. En la actualidad, existen algunos estándares que no utilizan la DCT, sino que utilizan la Transformada Wavelet y la codificación aritmética, como es el caso de JPEG2000 y MPEG-4 [1], [6].

Por otro lado, existe la *codificación temporal o Interframe*, la cual remueve las similitudes entre imágenes sucesivas, mediante la codificación de las diferencias entre cuadros sucesivos El estándar MPEG-2 realiza la compresión temporal analizando los cambios de un cuadro de video a otro mediante la observación del movimiento de macro bloques de 16x16 píxeles en las imágenes [6], [15].

Dentro del ámbito de la compresión de video se tiene el método de codificación *MJPEG (Motion Joint Photographic Experts Group)*, este método de compresión es utilizado en aplicaciones como cámaras de seguridad, transmisión de imágenes, y en áreas de identificación, medicina y fotografía, así como en algunos teléfonos celulares para grabar video, como el S8 de la marca SKYZEN. El método de compresión MJPEG consiste básicamente en manejar a la secuencia de imágenes que conforman el video de manera independiente, comprimiendo cada una mediante el algoritmo JPEG.

Las principales ventajas de MJPEG son que la compresión JPEG es muy factible de implementarse por hardware y soporta casi cualquier tamaño del video que se desea transmitir, por lo que es posible utilizar imágenes en tamaño *sub-QCIF (Quarter Common Intermediate Format)* hasta imágenes para *Televisión de Alta Definición (HDTV)*, al procesar las imágenes de manera independiente, si se presenta un error en una imagen, éste no se propaga en las siguientes imágenes como en el caso de MPEG-1 y MPEG-2. Otra de las ventajas de MJPEG que lo hace muy atractivo para sistemas de transmisión de video, es que conforme se va adquiriendo una imagen correspondiente a una secuencia de video, se puede comprimir para transmitirse inmediatamente, lo que reduce los tiempos de ejecución para todo el sistema.

Cuando se requiere que los procesos se ejecuten en tiempo real, es necesario realizarlos dentro del tiempo de muestreo de la señal, sin embargo, lograr esta aproximación tiene sus dificultades, ya que se requiere optimizar tanto la parte algorítmica como el software y contar con arquitecturas de procesamiento de alto desempeño. Las arquitecturas de los *procesadores digitales de señales (DSPs)* actuales pueden proporcionar una solución conveniente que combina flexibilidad y potencia computacional. El diseño de la familia de DSP's TMS320C6xxx (C6xxx) de la marca Texas Instruments está orientada hacia el procesamiento digital de imágenes.

En particular el DSP TMS320C6416 de punto fijo de alto desempeño (versión de 1Ghz) está basado en la tecnología VelociTI avanzado (velociTI.2), con la arquitectura de *palabra de instrucción muy larga (VLIW)* desarrollado por *Texas Instrument (TI)*. Este DSP ejecuta *8,000 millones de instrucciones por segundo (MIPS)* con una frecuencia de reloj de 1 GHz, además puede realizar ocho instrucciones de 32 bits/ciclo y 28 operaciones/ciclo [10].

Por otra parte, en el caso de adquisición de imágenes, en la actualidad los sensores tipo *CMOS (Semiconductor Complementario de Oxido-Metal)* han ganado mucho espacio en el mercado, debido a su bajo costo, con respecto a los sensores de tipo *CCD (Dispositivos acoplados por carga)*, además de que consumen poca potencia. Los sensores de video que existen actualmente en el mercado ya integran el proceso de conversión analógico digital, filtrado y cuantización por lo que entregan el video de manera digital en sus componentes *R, G, B (Rojo, Verde, Azul)* o bien en componentes *Y, Cr, Cb (Luminancia, Crominancia Roja, Crominancia Azul)*.

*El **objetivo** de este trabajo es implementar e integrar un sistema de adquisición y compresión de video utilizando los algoritmos de MJPEG, mediante una cámara de video digital tipo CMOS y una tarjeta de desarrollo del DSP C6416 (DSK6416), aprovechando el desempeño de la arquitectura del DSP para obtener tiempos de ejecución lo mas cerca posible al tiempo real. Posteriormente, como una etapa proyectada a futuro, el video comprimido se debe transmitir a una PC, ya sea a través de un puerto serie o por el puerto TCP/IP, para que ésta lleve a cabo la descompresión y visualización del video en pantalla.*

La implementación del sistema consiste de cinco etapas generales: la *adquisición*, la *compresión MJPEG*, la *transmisión de los datos comprimidos (etapa a futuro)*, la *descompresión MJPEG* y la *visualización de video*.

- En la *adquisición*, un sensor de video es el encargado de adquirir una secuencia de imágenes en formato digital y almacenarlas en un buffer de video. Posteriormente, con el DSP se realizan las funciones de submuestreo 4:2:0 y ordenamiento de los componentes de color Y, Cr, Cb de forma separada.

- En la etapa de *compresión*, se aplica el método de compresión MJPEG a las imágenes almacenadas en el buffer de video.

- En la etapa de *transmisión*, la cual esta planeada a futuro y fuera del alcance de este proyecto, los datos de video comprimido serían enviados a una PC a través del puerto serie RS-232.

- El proceso de *descompresión* es inverso al proceso de compresión y su finalidad es reconstruir el video original, a partir de los datos de video comprimido. Este proceso es ejecutado por el DSP. Los datos del video reconstruido son almacenados en un archivo para su visualización.

- Por ultimo, en la parte de *visualización* se despliega el video recuperado en el monitor de una PC.

El desempeño del sistema se evalúa con los tiempos de adquisición de imágenes, el porcentaje de compresión, los tiempos de procesamiento y visualización para una secuencia de video, la calidad visual y el porcentaje de error, con el fin de valorar la implementación del sistema en tiempo real utilizando este tipo de arquitectura.

La realización de la adquisición, compresión, descompresión y visualización del video sirven de base para la implementación a futuro de un sistema integral donde se pretende capturar video, transmitirlo y visualizarlo en tiempo real, haciendo uso de la compresión de video MJPEG.

Este trabajo presenta la siguiente estructura:

En el *primer capítulo* se describen los conceptos fundamentales de las imágenes y el video digital.

En el *segundo capítulo* se describen de manera general las técnicas de compresión de video digital, así como algunos de los principales estándares como MPEG1 y MPEG2, entre otros.

En el *tercer capítulo* se realiza una descripción detallada del método de compresión utilizado en este trabajo que es el MJPEG.

En el *cuarto capítulo* se expone como se realizó la implementación de la adquisición de video, utilizando una cámara de video digital y la arquitectura del DSP.

En el *quinto capítulo* se muestra la implementación de las etapas de compresión, descompresión y visualización de video, utilizando el DSP y la PC.

En el sexto *capítulo* se muestran los resultados obtenidos en la adquisición y compresión de video MJPEG, evaluando los diferentes grados de compresión aplicados a las secuencias de video, así como el análisis de los tiempos de procesamiento del sistema.

Además, se incluyen las secciones de *Conclusiones*, *Anexos*, *Glosario* y *Bibliografía*.

1.- FUNDAMENTOS DE VIDEO DIGITAL

Este capítulo tiene como objetivo describir los conceptos fundamentales del video digital, partiendo de las imágenes digitales, ya que éstas son sus elementos principales. Además, se analizan los espacios de color más comunes en video digital, y se describen los procesos de captura y despliegue de video digital, así como sus formatos y estándares. La comprensión de estos conceptos es de vital importancia para lograr los objetivos de desarrollo e implementación de nuestro sistema de adquisición y compresión de video digital.

1.1 Imágenes digitales

Una ***imagen*** es una fotografía de una escena en un instante particular en el tiempo, mientras que una ***secuencia de video*** representa la escena sobre un periodo de tiempo [1]. Un método común para definir a una imagen ***I*** es representarla como una matriz rectangular (llamada matriz de la imagen) como se muestra en la ecuación (1.1) [2].

$$I = [s(x, y)] \tag{1.1}$$

El valor de una fila, junto con el valor de una columna, define una pequeña área de la imagen llamada ***pixel*** (por las siglas *Picture Element*), el cual contiene un valor que representa el brillo o el color del pixel. En la figura 1.1 se observa una sección amplificada de una imagen, donde se aprecia un conjunto de pixeles que forman parte de ésta.

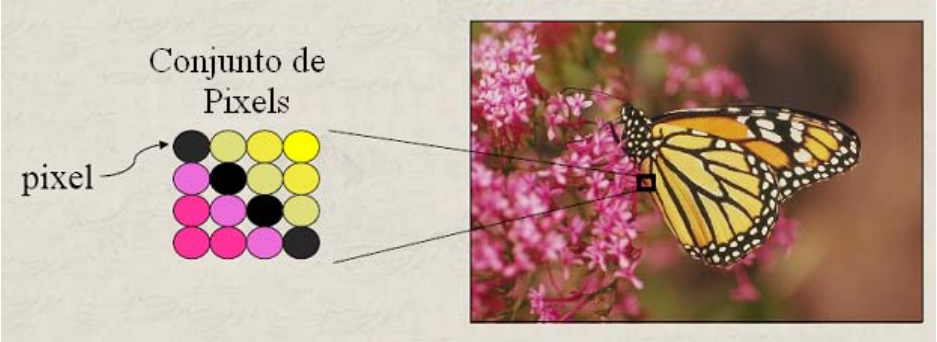


Figura 1.1 Conjunto de pixeles de una sección de una imagen.

La ***profundidad de bits*** es el número de bits que se le asigna a un pixel para representar la información de la imagen. Esto quiere decir que con una mayor profundidad de bits la imagen contendrá más información y por lo tanto se puede tener un mayor número de tonos y colores.

- Si se tiene una profundidad de un solo bit, solo se pueden tener dos *niveles* o *tonos*. En este caso tenemos una *imagen binaria* que consta únicamente de valores 0 y 1 (blanco y negro).
- Si se tiene una profundidad de ocho bits, es posible tener 256 *niveles* o *tonos*.

Los *niveles o tonos* que puede contener una imagen se encuentran mediante la relación $\mathbf{2^L}$, siendo *L* el número de profundidad de bits; para el caso de imágenes en tono real se tiene generalmente una profundidad de 24 bits, con lo que es posible generar 16,777,216 colores.

Una posibilidad para la representación de una imagen *s(x,y)* es asignarle valores pertenecientes al conjunto de escala de grises *G={0,1,…,255}*. El valor 0 de la escala de grises corresponde al negro y el 255 al blanco [2].

Imágenes a color

Típicamente, una imagen a color no puede ser representada por el modelo mostrado en la ecuación (1.1). En ese caso, se puede extender la imagen *I* usando más planos, por lo que se puede hablar de una imagen multiplanos, como se define en la ecuación (1.2) [2].

$$I = [s(x, y, n)] \tag{1.2}$$

donde ***n*** es el contador de planos.

La figura 1.2 muestra una imagen utilizando tres planos para almacenar la información de color; el color *rojo* es asignado al plano 0, el *verde* al plano 1, y el *azul* al plano 2. Un pixel de una imagen de *n* planos, es representado por un vector de *n* dimensiones donde los componentes g_n son elementos del conjunto de escala de grises *G*, como se ve en la ecuación (1.3) [2].

$$\bar{s}(x, y) = \begin{pmatrix} g_0 \\ g_1 \\ . \\ . \\ g_{n-1} \end{pmatrix} \tag{1.3}$$

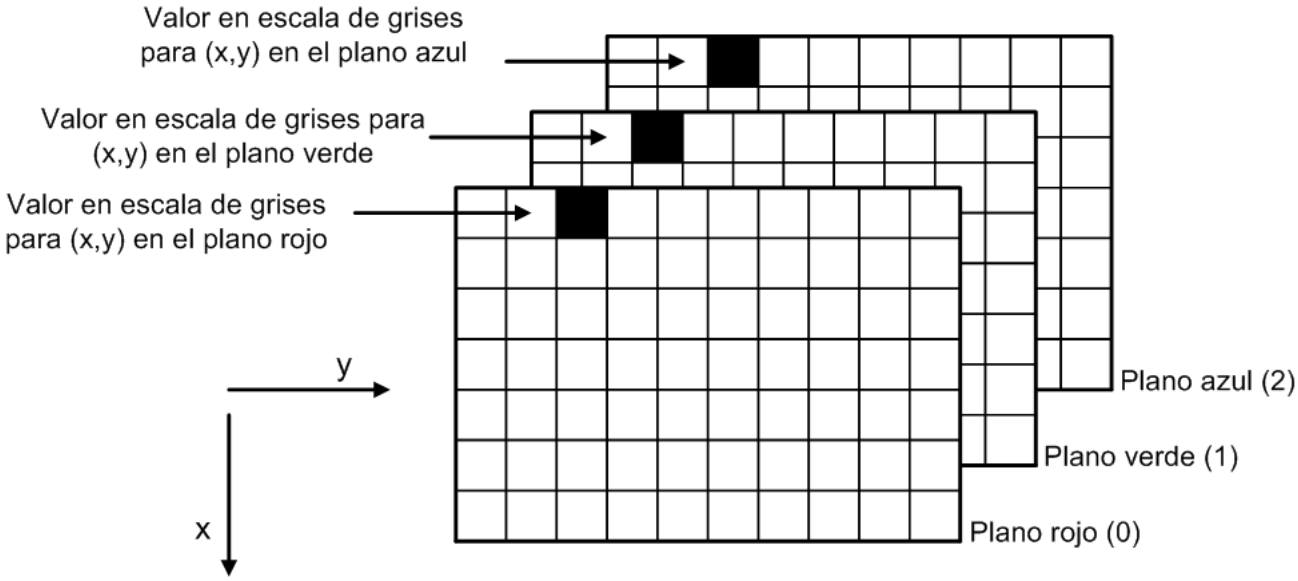


Figura 1.2 Definición de una imagen a color como una imagen de tres planos.

En la figura 1.2 se muestra la posición y orientación de los índices de las filas y columnas. El punto de inicio para ambos es la esquina superior izquierda de la imagen.

Cuando una imagen se despliega sobre la pantalla, cada elemento generalmente es mapeado a un pixel único de la pantalla. En la figura 1.3 se muestra la representación de un vector de un pixel en la memoria de la computadora, y donde se puede observar como la combinación de los valores de los componentes R, G, B nos dan como resultado un color final de pixel en la imagen.

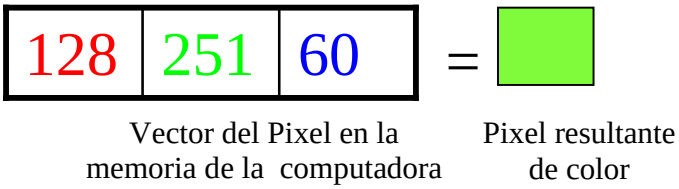


Figura 1.3 Representación de un pixel en la memoria de la computadora, y el pixel resultante.

Tipos de Imágenes Digitales.

Las imágenes digitales pueden ser divididas en dos clases generales, imágenes vectoriales e Imágenes de mapa de bits, las cuales se describen a continuación.

1.1.1 Imágenes Vectoriales.

Las ***imágenes vectoriales*** son generadas a partir de una serie de comandos de dibujo definidos por ecuaciones matemáticas para representar una imagen [3]. En la figura 1.4 se muestran algunas imágenes de este tipo.

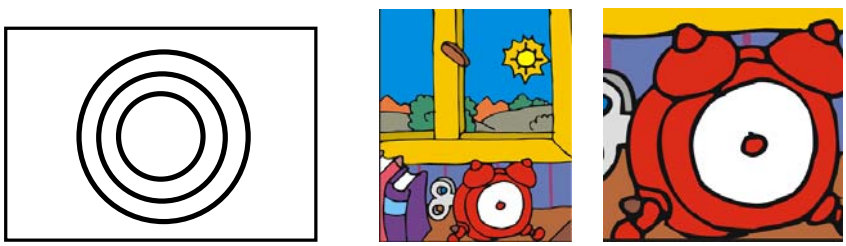


Figura 1.4 Imágenes Vectoriales.

Ventajas:

- Requieren poca memoria y poco espacio de almacenamiento.
- No pierden calidad al ser escalados, rotados o deformados. La figura 1.4b muestra una imagen vectorial y la figura 1.4c es una sección amplificada, donde se aprecia que la imagen original no sufre distorsión al ser amplificada.
- Fáciles de editar y modificar.

Desventajas:

- No son aptas para mostrar fotografías o imágenes muy complejas.

- Los datos que describen una imagen vectorial deben ser procesados. Si hay demasiados datos se puede hacer mas lento el proceso de despliegue.

- Para su visualización tanto en pantalla, como en la mayoría de sistemas de impresión, cada *vector* del objeto tiene que ser convertido a un *pixel* de la imagen [3].

1.1.2 Imágenes de Mapa de Bits

Las ***imágenes de mapa de bits*** *(bitmaps o imágenes raster)* son imágenes compuestas de pixeles y se representan como matrices de 2 dimensiones donde cada elemento de la matriz (pixel) representa un color o tono de gris que se mostrará en una localización específica de la pantalla [3]. En la figura 1.5 se muestra una imagen de mapa de bits.

Figura 1.5 Imagen de mapa de bits

Tamaño de una imagen.

El *tamaño de una imagen* en bytes se muestra en la ecuación (1.4) [3].

$$\frac{\text{Num. de pixeles por línea x Num. de líneas x bits por pixel}}{8} \qquad (1.4)$$

Resolución

La *resolución* de una imagen se define como el número de pixeles por unidad de área. A mayor número de pixeles por unidad de área, mayor será la resolución y menos imperfecciones se observarán en la imagen.

Ventajas:

- Son aptas para mostrar fotografías o imágenes complejas, de una calidad muy alta.

Desventajas:

- Requieren más memoria y espacio de almacenamiento que una imagen vectorial.

- Son dependientes del tamaño y no son adecuadas para una edición extensa. El aumento de tamaño ocasiona pérdida de calidad, como se ve en la figura 1.6 [3].

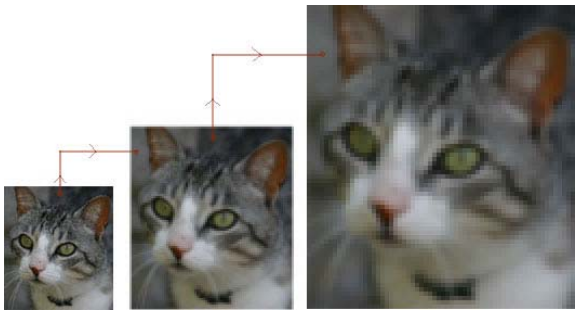


Figura 1.6 Imagen de mapa de bits donde se muestra su distorsión al ser amplificada.

1.2 El sistema visual humano

Un objetivo fundamental para el diseño de un sistema de visualización digital es que la imagen visual producida por el sistema pueda ser lo más parecido a la original y lo más agradable para el observador. Para poder lograr este objetivo es necesario tener en cuenta la respuesta del *sistema visual humano*. A través de este sistema, el ser humano puede observar, interpretar y responder a estímulos visuales.

La operación del Sistema Visual Humano es un área de estudio muy grande y compleja, y una profundización sobre el tema, esta fuera de este estudio. Algunas de las características más importantes del ojo humano, que tienen relevancia para el diseño de sistemas de video digital se describen en la tabla 1.1 [1].

Características del Sistema Visual Humano	Importancia para los sistemas de video digital
El ojo humano ante una sucesión rápida de imágenes tiene la percepción de un movimiento continuo.	Los cuadros (imágenes) que componen una secuencia de video, son presentados a una tasa lo suficientemente alta, de tal manera que el sistema visual humano al integrar esta secuencia de cuadros tenga la sensación de movimiento continuo.
El sistema visual humano retiene una imagen por una fracción de segundo, después de que la visualiza.	
El ojo humano es más sensible a la luminancia que al color (crominancia).	La resolución de color (o crominancia) puede ser reducida sin afectar de forma significativa la calidad de la imagen.
El ojo humano es mas sensible al alto contraste (diferencias grandes en luminancia) que al bajo contraste.	Los cambios grandes en luminancia (por ejemplo los bordes en una imagen) son particularmente importantes para la apariencia de la imagen.
La visión humana es mas sensible a frecuencias espaciales bajas (cambios en luminancia que ocurren sobre un área grande) que a frecuencias espaciales altas (cambios rápidos que ocurren sobre un área pequeña).	Es posible comprimir imágenes desechando algunas de las frecuencias altas menos importantes (sin embargo, la información de los bordes debe preservarse).

Tabla 1.1 Características del Sistema Visual Humano y su importancia en los sistemas de video digital.

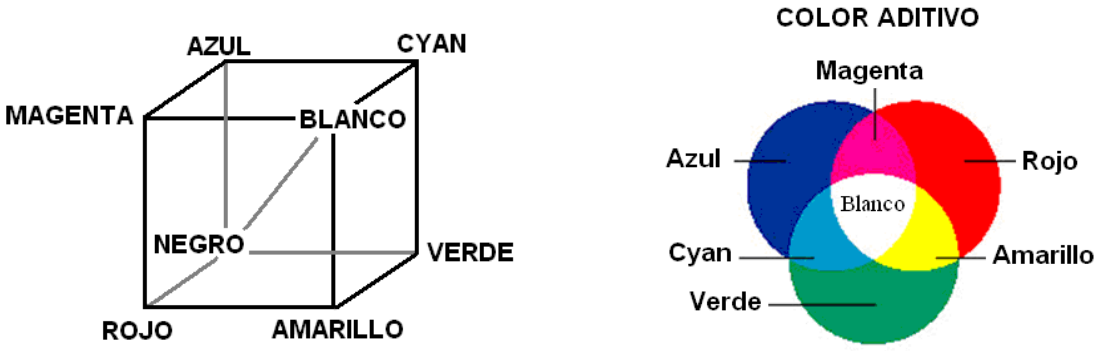
1.3 Espacios de color

*Un **espacio de color** es una representación matemática de un conjunto de colores.* Los modelos de color mas populares son el *Rojo, Verde, Azul (RGB),* utilizado en gráficos de computadora; los modelos de *Luminancia, Componente en fase y componente en Cuadratura (YIQ), Luminancia o Intensidad, Componente U y componente V (YUV), o Luminancia, Crominancia Roja y Crominancia Azul (YCbCr)* utilizados en sistemas de video; y el *Cyan, Magenta, Amarillo y Negro (CMYK),* utilizado en impresiones a color. Sin embargo, ninguno de estos espacios de color está directamente relacionado con las nociones intuitivas de matiz, saturación y brillo. Esto ocasionó la búsqueda de otros modelos, tales como el de *Matiz, Saturación e Intensidad (HSI)* y el *Matiz, Saturación y Valor (HSV),* los cuales simplificaran la programación, procesamiento y manipulación de los usuarios finales [10].

Todos los espacios de color se pueden derivar de la información RGB proporcionada por dispositivos de captura como cámaras y scaners. En este estudio nos concentraremos específicamente en tres de los espacios de color más comunes para representación de imágenes y video digital: *RGB, YUV y YCrCb.*

1.3.1 Espacio RGB

En el espacio de color RGB (Rojo, Verde y Azul), cada pixel está representado por tres números los cuales indican las proporciones del pixel con respecto al color rojo, verde y azul. Estos son los tres colores primarios aditivos de la luz y son representados por un sistema de coordenadas cartesiano tridimensional o cubo, como se muestra en la figura 1.7a. En la figura, el punto origen (R=0, G=0, B=0) corresponde al negro y el punto (R=1, G=1, B=1) al blanco. Cualquier color puede ser reproducido combinando varias porciones de rojo, verde y azul, de aquí surgen tres colores secundarios: el cian, magenta y amarillo, a esto también se le conoce como color aditivo, porque se obtiene de la suma de varios colores, como se observa en la figura 1.7b [1].



(a) (b)

Figura 1.7 Representación de color RGB.

Debido a que los tres componentes tienen la misma importancia para obtener el color final, los sistemas RGB usualmente representan cada componente con la misma precisión (y por ende el mismo número de bits). El uso de 8 bits por componente es bastante común, es decir, 3x8=24 bits, que son los requeridos para representar cada pixel. La figura 1.8 muestra una imagen a color y sus componentes R, G, B.

Imagen Original | Componente ***R*** | Componente ***G*** | Componente ***B***

Figura 1.8 Imagen original y sus componentes RGB.

1.3.2 Espacio YUV

El espacio de color YUV es utilizado por los estándares de video compuesto de Línea de Alternación por Fase - *Phase Alternation Line (PAL)*, el estándar del Comité de Sistemas de Televisión Nacional – *National Television System Commitee (NTSC)*, y el estándar de Color Secuencial con Memoria - *Sequentiel Couleur Avec Mémoire o Sequential Color with Memory (SECAM)*. En los inicios de la televisión, los sistemas en blanco y negro solo utilizaban información de luminancia (Y); la información del color (U y V) fue adicionada de tal manera que un receptor blanco y negro pudiera desplegar imágenes en blanco y negro de forma normal, y los receptores a color pudieran decodificar la información de color adicional para desplegar imágenes a color [5].

Las ecuaciones básicas para convertir una imagen del espacio RGB al YUV, de acuerdo con la ecuación (1.5), son [6]:

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ V &= R - Y \\ U &= B - Y \end{aligned} \qquad (1.5)$$

donde: Y es la luminancia, U y V son los componentes de color.

La figura 1.9 muestra una imagen a color y sus componentes separados en el espacio YUV.

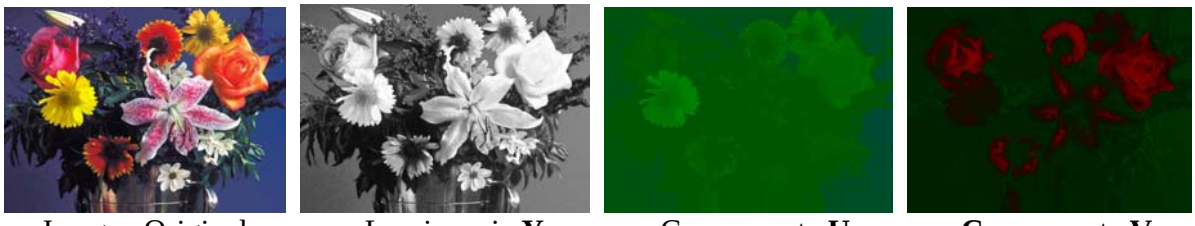


Imagen Original | Luminancia **Y** | Componente **U** | **C**omponente **V**

Figura 1.9 Imagen original y sus componentes YUV.

Como se observa en la figura 1.9, el componente de luminancia conserva la mayoría de rasgos a nivel de tonos de gris de la imagen original.

1.3.3 Espacio YCrCb

Un espacio de color muy popular de este tipo es el *YCrCb*. ***Y*** es el componente de luminancia, esto es, una versión monocromática del color de la imagen. ***Y*** es un promedio entre **R**, **G** y **B**, según el estándar ITU CCR 601, como se describe en la ecuación (1.6) [1]:

$$Y = 0.299R + 0.587G + 0.114B \tag{1.6}$$

La información de color puede ser representada como una ***diferencia de color*** o ***componentes de crominancia***, donde cada componente de *crominancia* es la diferencia entre R, G o B y la *luminancia* ***Y***, según la ecuación (1.7) [1]:

$$\begin{aligned} C_r &= R - Y \\ C_b &= B - Y \\ C_g &= G - Y \end{aligned} \tag{1.7}$$

La descripción completa está dada por *Y (luminancia)* y *tres diferencias de color Cr, Cb y Cg* que representan la variación entre la intensidad de color y el fondo de luminancia de la imagen. Esta representación tiene poco mérito con respecto al espacio RGB; ahora tenemos cuatro componentes en lugar de tres. Sin embargo, resulta que el valor de Cr + Cb + Cg es una constante. Esto significa que solo se necesitan dos de los tres componentes de crominancia para ser transmitidas: el tercer componente se puede reconstruir a partir de los otros dos.

En la ecuación 1.6 se observa que el componente *G* es el que aporta más información a la luminancia, por lo tanto en la ecuación (1.7) el componente *Cg* sería más pequeño que los otros dos componentes Cr y Cb. *En el espacio YCrCb, únicamente son transmitidas la luminancia (Y) y la crominancia roja y azul (Cr, Cb).* El componente de crominancia verde *Cg*, no se transmite debido a que al ser más pequeño, es más susceptible al ruido [1].

Para convertir una imagen *RGB* dentro del espacio de color *YCrCb* y viceversa se utilizan las ecuaciones (1.8) y (1.9). Note que *G* puede ser extraído de la representación *YCrCb* sustrayendo *Cr* y *Cb* de *Y* [1].

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ Cb &= 0.564(B - Y) \\ Cr &= 0.713(R - Y) \end{aligned} \tag{1.8}$$

$$\begin{aligned} R &= Y + 1.402Cr \\ G &= Y - 0.344Cb - 0.714Cr \\ B &= Y + 1.772Cb \end{aligned} \tag{1.9}$$

La figura 1.10 muestra una imagen a color y sus componentes separados en el espacio *YCrCb*.

Imagen Original | Componente ***Y*** | Componente ***Cr*** | Componente ***Cb***

Figura 1.10 Imagen original y sus componentes *YCrCb*.

La ventaja clave de *YCrCb* sobre *RGB* es que los componentes *Cr* y *Cb* pueden ser representados con una resolución más baja que *Y* debido a que el ojo humano es menos sensible al color que a la luminancia. Esto reduce la cantidad de datos requeridos para representar los componentes de crominancia sin que se tenga un efecto significativo en la calidad visual: para un observador casual, no hay diferencia aparente entre una imagen *RGB* y una imagen *YCrCb* con la resolución de crominancia reducida. En la *sección 1.5.2* se profundiza más sobre este tema.

Aplicaciones de los espacios de color.

- RGB: Gráficos de computadora a color, TV a color, imágenes a color
- YUV: Estándares de video compuesto PAL, NTSC y SECAM
- YCrCb: Transmisión de televisión, esquemas JPEG.

El procesamiento de video requiere algunas veces que se alterne de un espacio de color a otro, como es el caso de sistemas de transmisión de video, donde se transmite utilizando el espacio de color *YUV* o *YCrCb* y se despliega la imagen utilizando el espacio de color *RGB*.

1.4 Sensores de imágenes y video

Los sensores de imágenes y video son dispositivos que perciben las variaciones de intensidad de la luz, pero sin distinguir los colores de la imagen; para que el sensor pueda captarlos, se emplean filtros que separan los colores de la escena en Rojo, Verde y Azul.

Un sensor digital de imágenes es una matriz de pequeñas celdas perfectamente alineadas en filas y/o columnas. Cada una de esas celdas es un elemento fotosensible microscópico, con la capacidad de generar electrones en función de la cantidad de luz que recibe. Cada celda producirá un flujo eléctrico variable sobre la base de cantidad de luz que incida en su superficie. Actualmente destacan dos tipos de sensores digitales de cámaras de vídeo o de fotografía: los *Dispositivos Acoplados por Carga (CCD)* y los de tipo *Semiconductor Complementario de Oxido-Metal (CMOS).*

1.4.1 Sensores tipo CCD

El funcionamiento de los sensores CCD se basa en la estructura de un *Transistor de Efecto de Campo de Semiconductor-Oxido-Metal (MOSFET)* que se representa en la figura 1.11.

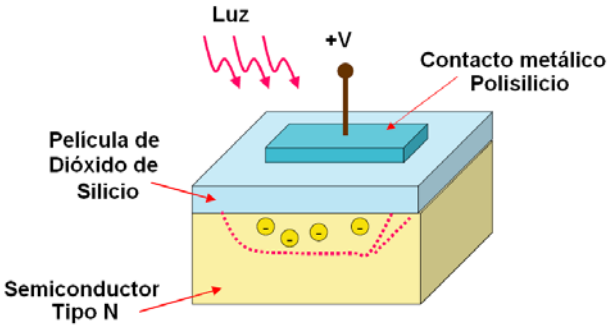


Figura 1.11 Diagrama básico de un semiconductor MOSFET usado en células CCD

El electrodo de polisilicio tiene propiedades metálicas, pero, a diferencia de los MOSFET convencionales, es transparente permitiendo el paso de la luz y por lo tanto la generación de un flujo eléctrico a partir de los fotones incidentes sobre el semiconductor. El electrodo metálico está aislado del semiconductor tipo P mediante una película de dióxido de silicio. Si se aplica un voltaje positivo al electrodo, gran parte de los electrones y huecos generados a partir de la incidencia de un fotón se concentrarán, por atracción eléctrica, bajo el electrodo positivo. Si no existiera este voltaje positivo los electrones y huecos se recombinarían dando lugar a una carga neta nula. En consecuencia, la carga almacenada debajo del contacto de polisilicio es proporcional a la cantidad de luz incidente sobre la superficie del electrodo [4].

CCD de línea

La estructura básica de una célula CCD es una cadena de semiconductores MOSFET, con un sustrato tipo N común a todos ellos y situados lo suficientemente próximos para que pueda realizarse una interacción entre las cargas a partir del control electrónico de los voltajes aplicados a las terminales. Es importante observar que la carga almacenada es un valor analógico que se debe extraer de la estructura CCD para obtener su valor. El control electrónico de un sensor de imagen CCD se realiza en dos fases: adquisición de imagen y lectura de datos. Los voltajes aplicados a los electrodos durante la fase de adquisición de imagen se representan en la figura 1.12 para una célula CCD de tres fases [4].

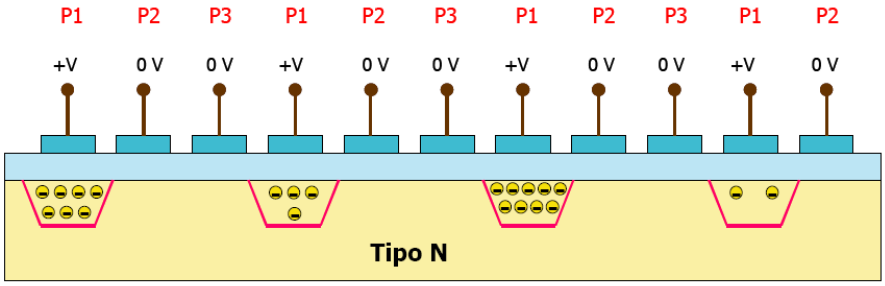


Fig. 1.12 Estructura CCD durante la fase de adquisición de imagen.

En esta configuración se aplica un voltaje positivo a una de cada tres terminales, de modo que las cargas acumuladas quedan registradas bajo los contactos de estos electrodos. El tiempo durante el que se aplican las condiciones de adquisición de la imagen es equivalente al tiempo de exposición del sensor, ya que la carga total almacenada será proporcional al valor medio de la luz incidente durante este periodo de tiempo.

Una vez registrada la luz incidente en la estructura CCD se debe proceder a su lectura. El proceso de lectura se realiza modificando los voltajes aplicados sobre los terminales P1, P2 y P3, de forma que externamente se provoque una transferencia de las cargas entre los MOSFETs adyacentes.

El desplazamiento de la carga se obtiene aplicando señales de reloj externas a los electrodos de la estructura. En la fase de adquisición la carga se encuentra bajo los contactos P1, que están a un voltaje positivo. Para desplazar la carga al contacto P2 se aplica un voltaje positivo a este electrodo, manteniendo constante la tensión del electrodo P1. Bajo estas condiciones, la carga almacenada bajo el contacto P1 se reparte ahora entre los contactos P1 y P2, si ahora bajamos la tensión del contacto P1, toda la carga se situará bajo el terminal P2. De modo análogo, es posible desplazar la carga de P2 a P3 y de P3 a P1, desplazándose todas las cargas hacia la derecha hasta que son leídas en la célula de terminación de la estructura. Normalmente, las señales de reloj para la lectura se proporcionan desde un circuito integrado externo. En la figura 1.13 se representa un diagrama de las formas de onda que deben aplicarse a los terminales P1, P2 y P3 para el desplazamiento de las cargas [4].

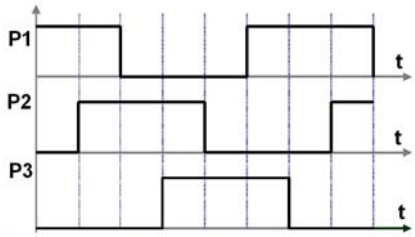


Figura 1.13 Diagrama de las señales de lectura en una CCD de tres fases.

La configuración CCD vista hasta este momento se conoce como *CCD de línea*, debido a que los sensores están situados uno al lado del otro formando una línea recta. Este tipo de estructuras se utilizan en scanners, fotocopiadoras, faxes y las denominadas cámaras lineales, que se utilizan en algunos sistemas de visión industrial por computadora.

Para obtener una imagen con este tipo de dispositivos es necesario que exista un desplazamiento relativo entre el objeto y el sensor CCD. La imagen se va explorando línea a línea y se reconstruye a partir del movimiento relativo existente entre el sensor y el objeto.

Uno de los problemas inherentes al procedimiento de lectura del sensor CCD es que si la estructura se mantiene expuesta a la luz mientras se van desplazando las cargas, se va añadiendo una carga remanente a medida que éstas se desplazan hacia la terminal de lectura. El efecto es particularmente notorio cuando existe un punto de luz de gran intensidad. En este caso, todas los elementos situados a la izquierda de este punto de luz (suponemos que las cargas se desplazan hacia la derecha) adquirirán una carga adicional cuando pasen por debajo del contacto expuesto a una gran intensidad lumínica. El efecto se conoce con el nombre de *smear* donde se percibe visualmente como el punto más brillante deja una estela de luz en todos los elementos de imagen situados a su izquierda y en su misma línea [4].

CCD Matriciales.

En las cámaras de video actuales, los sensores CCD tienen una estructura matricial, de modo que puede capturarse toda la imagen de forma simultánea sin necesidad de desplazar el sensor o el objeto. Un sensor matricial está formado por una agrupación de elementos CCD lineales dispuestos en forma de una matriz, tal y como se ilustra en la figura 1.14.

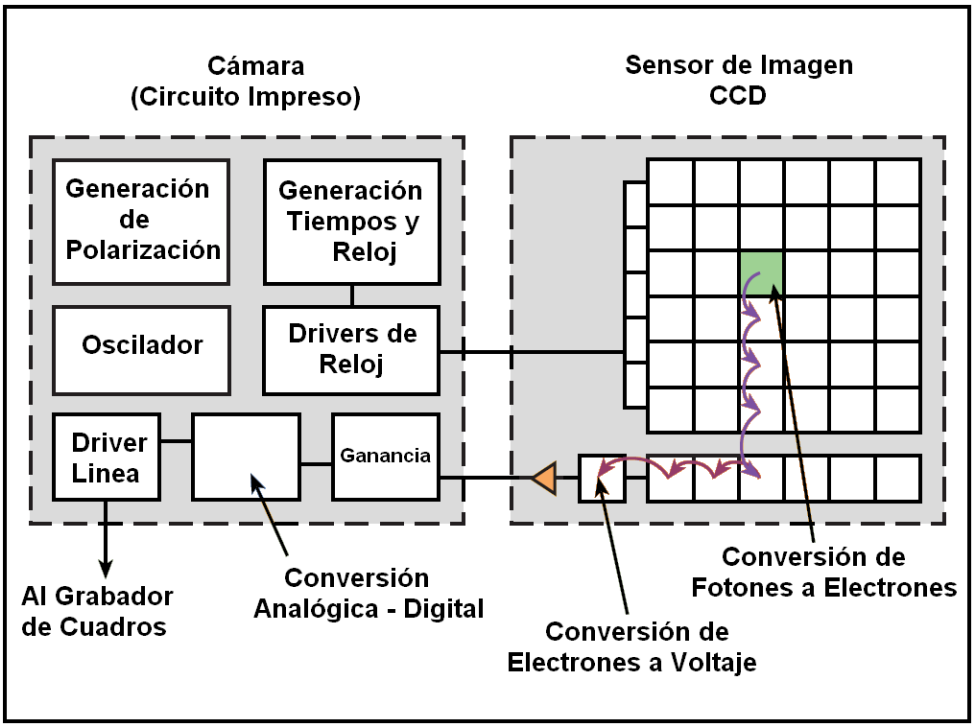


Figura 1.14 Diagrama a bloques de una cámara CCD.

Un grupo óptico, el cual está formado por varias lentes, se encarga de formar la imagen sobre la superficie del sensor de modo que todos los elementos de la imagen son adquiridos simultáneamente. Una vez capturada la imagen, se efectúa la lectura de la información desplazando la carga que posee un elemento de CCD de línea al elemento adyacente y así sucesivamente hasta llegar a un registro (también formado por dispositivos de carga acoplada) que es el encargado de ir suministrando, por orden secuencial, las diferentes cargas que poseen los distintos MOSFETs que forman el sensor. Estas cargas electrónicas se convierten en voltaje, el cual se amplifica y se recoge en el circuito integrado de la cámara, encargado de procesar estos datos y proporcionar una señal digital, la cual se grabará en memoria [4], [7].

1.4.2 Sensores de imágenes y video tipo CMOS

Estos dispositivos están basados en la tecnología CMOS *(Semiconductor Complementario de Oxido-Metal)*, la cual se basa en la utilización de dos transistores MOSFET, uno del *tipo n (NMOS)* y otro del tipo *p (PMOS)*, integrados en un chip único de silicio. Los dispositivos CMOS se caracterizan por una alta velocidad de acceso y un bajo consumo eléctrico.

Además, los sensores CMOS se caracterizan porque cada elemento de sensado contiene la electrónica necesaria para convertir la carga de electrones generada en voltaje, así como un amplificador de la señal y un registro individual para cada elemento de sensado, como se muestra en la figura 1.15. Esto proporciona la ventaja de que se puede acceder a la información captada en la totalidad del dispositivo y también en una zona en particular de éste [2], [7].

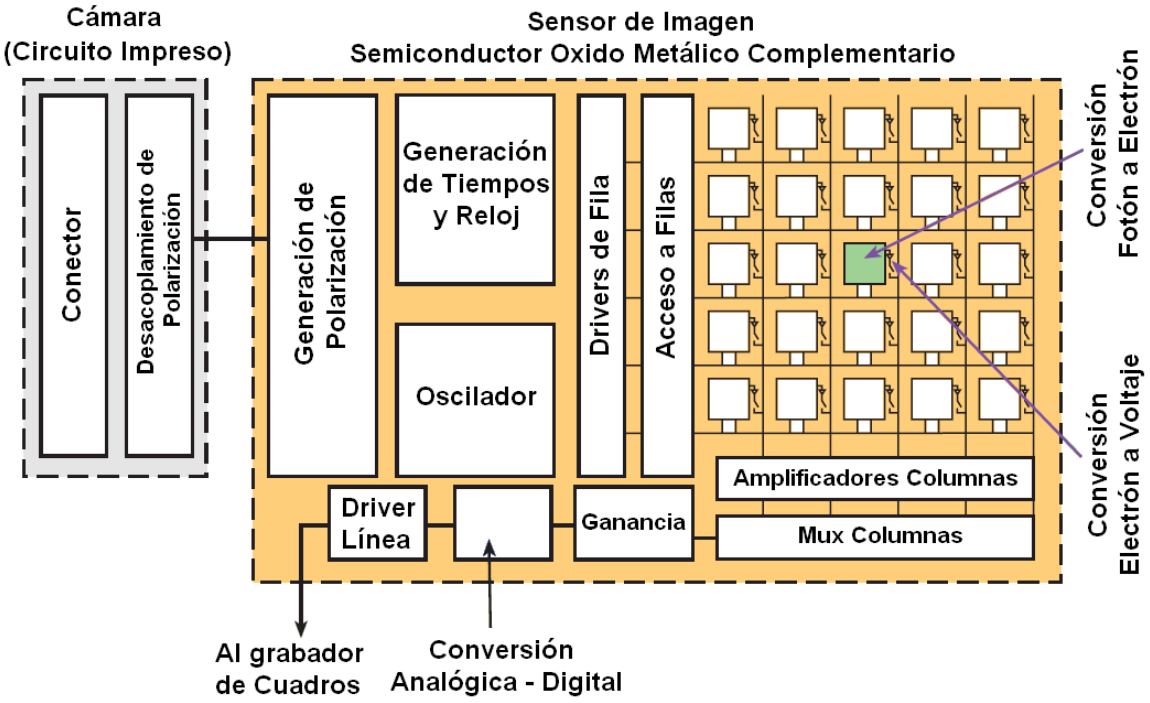


Figura 1.15 Diagrama a bloques de una cámara CMOS.

Gracias a la tecnología CMOS se pueden integrar tanto los elementos de sensado, como la electrónica necesaria para realizar las funciones de *control y lectura*, *ajuste de ganancia*, *conversión analógico-digital*, *oscilador*, *generadores de polarización*, *generador de tiempos y reloj*, lo que se refleja en un menor tamaño de los circuitos para la captura de imágenes.

En la tarjeta de circuito impreso integrado en la cámara, el cual es independiente del sensor, se realizan únicamente las funciones de *desacoplamiento de polarización* y de conexión externa [2], [7].

1.4.3 Ventajas y desventajas de los sensores de video CCD y CMOS.

Actualmente, existe una fuerte competencia entre las tecnologías de sensores de video CCD y CMOS. Ambos tienen ventajas y desventajas, dependiendo el tipo de aplicación que se requiera. Antes de hacer una comparación entre estos dos tipos de sensores, haremos una descripción de las características más importantes de los sensores de imágenes y video:

- **Respuesta:** Es la capacidad que tiene el sensor de generar carga eléctrica por unidad de luz que incide sobre él.

- **Intervalo Dinámico:** Indica el nivel de señal que es posible medir entre el umbral del fotodetector y su saturación, lo que influye en la gama de luminosidad que se puede obtener del sensor.
- **Uniformidad:** Es la consistencia de la *respuesta* para diferentes pixeles bajo condiciones idénticas de iluminación. Idealmente, el comportamiento debe ser uniforme, pero las variaciones espaciales en el procesamiento de la oblea, defectos en partículas y variaciones en el amplificador distorsionan la uniformidad.

- **Shuttering:** Es la habilidad de iniciar y detener la exposición de manera arbitraria.

- **Ventaneo:** Es la capacidad para leer una porción de la imagen detectada por el sensor. Esto permite elevar la tasa de cuadros o de líneas para las pequeñas regiones de interés.

- **Antiblooming:** *Blooming o smear*, es el efecto, por el cual si un elemento de sensado del dispositivo se satura puede afectar a otros elementos de sensado próximos a él. El *antiblooming* es la capacidad del dispositivo para drenar la sobre exposición de luz localizada, sin comprometer al resto de la imagen detectada.

Otras características importantes son: *la electrónica de control, la velocidad, polarización, el tipo de reloj, la confiabilidad del sensor, su inmunidad al ruido y el consumo de potencia.* En la tabla 1.2 se hace una comparación entre las cámaras de video CCD y CMOS [2], [4], [7].

Características	CCD	CMOS
Respuesta	Más rápida en CMOS que en CCD.	
Intervalo Dinámico	Mejor en CCD que en CMOS, por un factor aproximadamente de 2 a 1.	
Uniformidad	Mejor en CCD que en CMOS	
Shuttering	Mejor en CCD que en CMOS	
Ventaneo	Muy limitado o no soportado.	Si
Blooming	Puede reducirse	Muy bajo
Electrónica de Control y Reloj	Externos al sensor	Integrados con el sensor
Ruido	Más bajo en CCD que en CMOS, debido a su menor integración de componentes.	
Consumo de Potencia	CMOS consume mucho menos potencia que CCD.	
Costo	Alto, requiere varios chips.	Bajo, integrable en un chip único.

Tabla 1.2 Comparación entre sensores de video CCD y CMOS

De lo anterior, podemos concluir que los sensores CCD ofrecen mejor calidad de imagen, mayor inmunidad al ruido y mayor flexibilidad que los sensores CMOS, sin embargo estos últimos consumen mucho menos energía y permiten una mayor integración. Ambas tecnologías están evolucionando hacia mayores niveles de calidad, por lo que en un futuro próximo veremos cámaras que incorporan CMOS con un nivel de calidad equiparable a los mejores CCD, sin una diferencia de costo apreciable entre un dispositivo y otro.

1.5 Formatos de Video Digital

Aunque hay muchas variaciones y técnicas de implementación, las señales de video son solo una vía para la transferencia de información visual de un punto a otro. La información puede venir de un reproductor de DVD, de televisión abierta o por cable, de un sistema satelital, de Internet o de muchas otras fuentes.

1.5.1 Video Digital

Una escena visual real es continua tanto espacial como temporalmente. Para poder representar y procesar una escena visual digitalmente es necesario muestrear la escena real tanto espacialmente (típicamente sobre un mapa rectangular en el plano de la imagen de video) y temporalmente (típicamente como una serie de *imágenes* o *cuadros* (*frames*) muestreados en intervalos regulares de tiempo) como se muestra en la figura 1.16 [1].

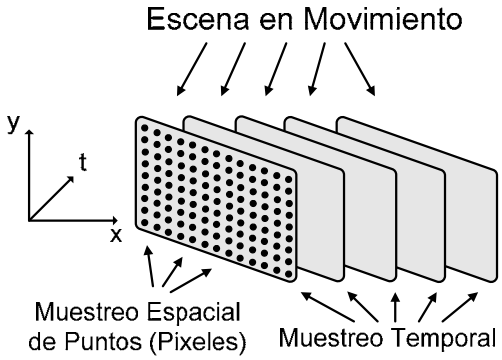


Figura 1.16 Muestreo espacial y temporal de una escena en movimiento.

Video Digital: Es la representación de una escena de video muestreada espacial y temporalmente en forma digital, es decir, una secuencia de imágenes digitales que cambian con respecto al tiempo, lo cual da una sensación de movimiento. Cada muestra espacio-temporal es representada digitalmente con un *pixel* (*sección 1.1*) [1].

En la figura 1.17 se muestra un sistema de video digital. A la entrada del sistema, se captura una escena visual real y se convierte en una representación digital muestreada. Esta señal de video puede ser manejada digitalmente para procesamiento, almacenamiento y transmisión. A la salida del sistema, la señal de video digital se reproduce en una pantalla para poder visualizarla.

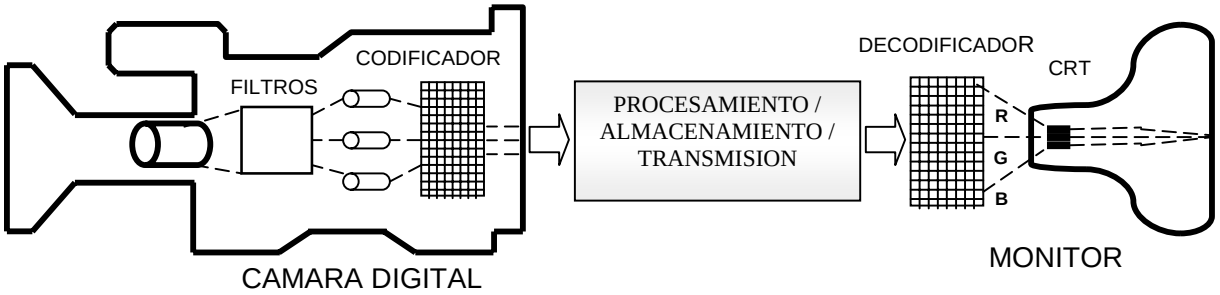


Figura 1.17 Sistema de video digital.

Para la generación y representación digital de una escena de video se pueden considerar dos etapas: adquisición y digitalización.

La digitalización puede realizarse de manera externa utilizando un dispositivo o una tarjeta de forma separada (por ejemplo una tarjeta de captura de video en una PC); cada vez es más común que el proceso de digitalización esté integrado con las cámaras, de tal manera que a la salida de la cámara se tiene una señal muestreada en formato digital. Las etapas de procesamiento y visualización se analizan más adelante.

1.5.2 Muestreo, submuestreo y tasa de cuadros.

Muestreo

Una imagen digital puede ser generada muestreando una señal de video analógico (esta es una señal eléctrica variable que representa una imagen de video) en intervalos regulares. El resultado es una versión muestreada de la imagen; la imagen muestreada está definida únicamente en una serie de puntos muestreados que están espaciados regularmente, como se ve en la figura 1.18.

La calidad visual de la imagen está determinada en gran medida por el número de puntos muestreados. Entre más puntos muestreados se tenga (resolución de muestreo alta) se tendrá una representación más fina de la imagen. Sin embargo, para una gran cantidad de puntos muestreados se requiere una alta capacidad de almacenamiento. En la *sección 1.5.3* se hace un análisis de las resoluciones de los formatos de video más comunes.

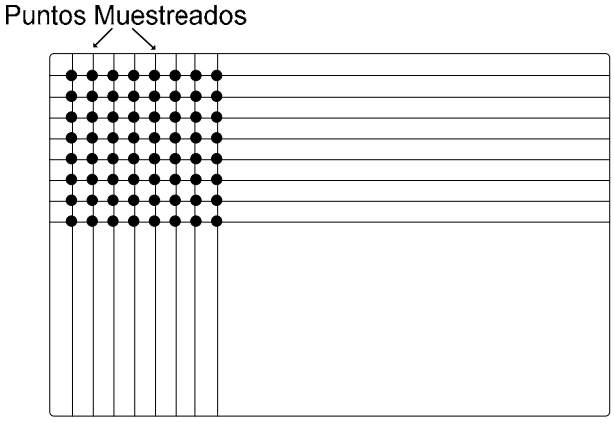


Figura 1.18 Muestreo de una imagen de video.

Submuestreo

El término *submuestreo* significa que algunos componentes (del espacio de color) de la señal de video son muestreados a una tasa más baja que otro componente de la misma señal en el sistema. El *submuestreo* se utiliza para reducir el ancho de banda requerido para la transmisión de video, además de que es ampliamente utilizado en la compresión de video, usando el espacio de color *YCrCb* [8].

La figura 1.19 muestra tres de los formatos de submuestreo más populares para Cr y Cb, los cuales se describen enseguida:

- 4:4:4 significa que los tres componentes (Y, Cr y Cb) tienen la misma resolución y, por lo tanto, existe una muestra por componente en cada posición de pixel. (Los números indican la tasa de muestreo relativa de cada componente en la dirección horizontal, esto significa que por cada 4 muestras de luminancia hay 4 muestras de Cr y 4 de Cb.) El muestreo 4:4:4 conserva totalmente la fidelidad de los componentes de crominancia [1].

- En el submuestreo 4:2:2, los componentes de crominancia tienen la misma resolución vertical pero tienen la mitad de la resolución horizontal (los números indican que por cada 4 muestras de luminancia en la dirección horizontal hay 2 muestras de Cr y 2 de Cb). El formato 4:2:2 es usado para la reproducción de color de alta calidad [1].

- 4:2:0 significa que tanto Cr como Cb tienen la mitad de la resolución horizontal y vertical de *Y*. El término 4:2:0 es un poco confuso: los números realmente no tienen una interpretación adecuada y parecen haber sido elegido históricamente como un "código" para la identificación de este formato de muestreo en particular. El formato de muestreo 4:2:0 es muy popular en aplicaciones como videoconferencias, televisión digital y almacenamiento en DVD. Debido a que cada componente de crominancia contiene un cuarto de las muestras del componente *Y*, el formato 4:2:0 requiere la mitad de muestras totales que el formato 4:4:4 (o que RGB) [1].

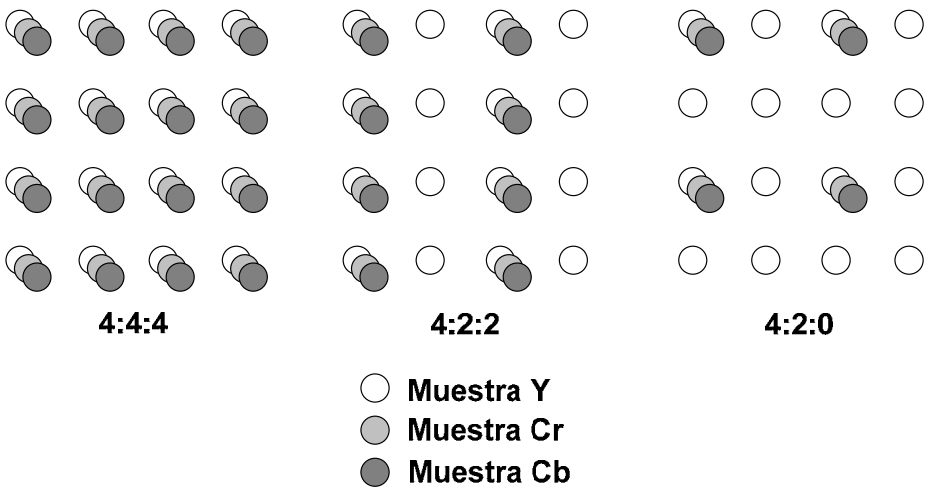


Figura 1.19 Formatos de submuestreo de crominancia

El formato de submuestreo 4:2:0 algunas veces es descrito como *"12 bits por pixel"*. La razón de esto puede ilustrarse examinando un grupo de 4 pixeles en la figura 1.20. El diagrama de la izquierda muestra un muestreo 4:4:4: se requiere un total de 12 muestras, 4 por cada componente (Y, Cr y Cb), siendo necesarios un total de 12 x 8 = 96 bits, lo que nos da un promedio de 96/4 = *24 bits por pixel*. El diagrama de la derecha muestra un muestreo 4:2:0, en donde se requiere de 6 muestras, de las cuales 4 son de Y, una corresponde a Cr y una a Cb, obteniendo un total de 6 x 8 = 48 bits, teniendo en promedio 48/4 = *12 bits por pixel* [1].

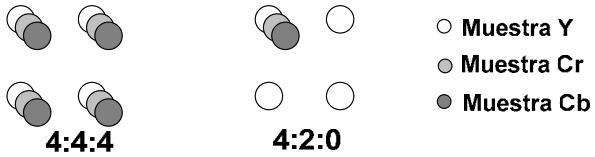


Figura 1.20 Cuatro pixeles: de 24 bits por pixel y 12 bits por pixel

Con el formato de submuestreo 4:2:2 la cantidad de información de las imágenes se reduce en un 33.33% conservándose la calidad, mientras que con 4:2:0 la información se reduce en un 50%, todo esto con respecto al formato 4:4:4.

Tasa de Cuadros

En un sistema de video es necesario saber cuántas imágenes por segundo se deben tener para que produzcan en el ojo humano una sensación de movimiento continuo, por lo que es importante definir los siguientes conceptos:

Tasa de Cuadros: Es el número de veces por segundo en la que un cuadro completo (imagen o frame) se despliega completamente en pantalla, usualmente 30 o 25 cuadros por segundo. En algunos casos la tasa de cuadros se define como la tasa a la cual son transmitidos los cuadros completos [8].

Una tasa de cuadros alta da una apariencia de movimiento más suave en las escenas de video, pero requiere que se capturen y almacenen una gran cantidad de muestras. Las tasas de cuadros debajo de 10 cuadros por segundo suelen ser utilizadas para comunicaciones de video con una tasa de bits muy baja (debido a que la cantidad de datos para esta tasa de cuadros es relativamente pequeña). Sin embargo, el movimiento es claramente abrupto y anormal con esta tasa. La tasa entre 10 y 20 cuadros por segundo es mas típica para comunicaciones de video con una tasa de bits baja; 25 o 30 cuadros por segundo es el estándar para la televisión; 50 o 60 cuadros por segundo es apropiado para video de alta calidad (a costa de una alta tasa de datos). La tabla 1.3 muestra la relación entre la tasa de cuadros de video y la apariencia que genera al observador [1].

Tasa de cuadros de video	Apariencia
Abajo de 10 cuadros por segundo.	Apariencia de movimiento anti-natural o entrecortado.
10 – 20 cuadros por segundo.	Si el movimiento de la escena es lento, la apariencia es buena. Si el movimiento es rápido, da la apariencia de estar entrecortado.
20 – 30 cuadros por segundo.	Buena percepción del movimiento de la escena.
50 – 60 cuadros por segundo.	Muy buena percepción del movimiento de la escena.

Tabla 1.3 Tasas de cuadros de video.

1.5.3 Formatos y estándares de Video Digital.

Formatos de video digital.

En el área de codificación de video, éste frecuentemente es convertido a algún "formato intermedio" antes de pasar a las etapas de compresión y transmisión. Un conjunto de formatos de cuadro muy popular es el que está basado en el formato CIF *(Formato Intermedio Común)*, en el cual cada cuadro tiene una resolución de 352 x 288 pixeles. La resolución de algunos de estos formatos se muestra en la tabla 1.4 y sus dimensiones relativas se ilustran en la figura 1.21. Todos estos formatos son *progresivos* (*sección 1.6*) y utilizan el *submuestreo 4:2:0* [1].

Formato	Resolución de Luminancia (Horizontal x Vertical)	Aplicaciones
Sub-QCIF	128 x 96	Aplicaciones multimedia móviles.
Quarter CIF(QCIF)	176 x 144	Aplicaciones multimedia móviles. Videoconferencias.
CIF	352 x 288	Videoconferencias.
4CIF	704 x 576	TV de Definición Estándar

Tabla 1.4 Formatos Intermedios CIF.

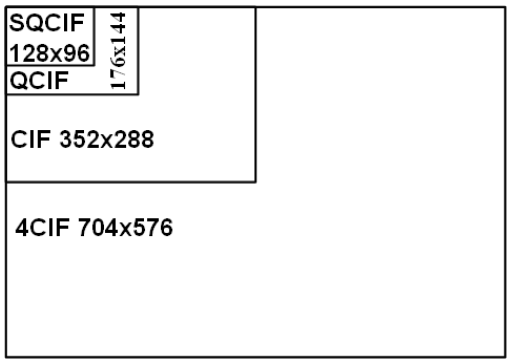


Figura 1.21 Formatos Intermedios CIF.

Otros formatos

Alrededor del mundo se emplean diferentes tamaños o formatos de video, dependiendo de la aplicación, como es el caso de los formatos SIF (*Formato de Entrada de Fuente*), los cuales suelen utilizarse en aplicaciones de almacenamiento o los formatos QVGA (240x320) y VGA (480x640) que se utilizan en las computadoras, los cuales se describen en la tabla 1.5 [6].

Formato	Resolución de Luminancia (Horizontal x Vertical)	Nombre
VGA	640x480	*Video graphics array*
SVGA	800x600	*Super video graphics array*
XVGA, XGA	1024x768	*Extended video graphics array*
SXGA	1280x1024	*Super XGA*
UXGA	1600x1024	*Ultra XGA*

Tabla 1.5 Resolución de Monitores de Computadora.

La ***relación de aspecto*** es el valor del cociente que resulta de dividir la resolución vertical (número de pixeles) entre la resolución horizontal (número de líneas). Alrededor del mundo se emplean diferentes relaciones dependiendo de la aplicación; por ejemplo, la relación 3:2 se usa en películas de 35 mm y la razón 16:9 suele utilizarse para televisión estándar y de alta definición.

Organizaciones de Estándares de Video

Las Organizaciones de Estándares de video fueron creadas para regular y normalizar la transmisión de video analógico, principalmente para televisión. Con el auge de la tecnología digital, estas organizaciones extendieron sus estándares al video digital. A continuación se describen las tres organizaciones más importantes en el área de estándares de video.

NTSC *(National Television Systems Committee)*
- Usado en Norteamérica, Japón, Taiwán, y partes del Caribe y Sudamérica.
- Utiliza 525 líneas de escaneo por cuadro, 29.97 cuadros/seg, relación de aspecto 4:3, espacio de color YUV y algunas veces YIQ, hace uso de video entrelazado *(En la sección 1.6 se profundiza mas en el concepto de video progresivo y entrelazado).*

PAL *(Phase Alternating Line)*
- Utilizado en Francia y otros países de Europa, Australia y Nueva Zelanda
- Utiliza 625 líneas de escaneo por cuadro, 25 cuadros/seg, relación de aspecto 4:3, espacio de color YUV, hace uso de campos entrelazados.

SECAM *(Système Electronique Couleur avec Mémoire)*
- Usado en Rusia y el este de Europa.
- Utiliza 625 líneas de escaneo por cuadro, 25 cuadros/seg, relación de aspecto 4:3, espacio de color YUV, hace uso de campos entrelazados (difiere del PAL en la codificación del color) [5], [6].

1.6 Despliegue de Video

La visualización de una señal de video implica la proyección de cada cuadro de video sobre una pantalla o monitor. Existen dos métodos de escaneo muy utilizados para el despliegue de imágenes de video: *el método de escaneo entrelazado y el progresivo.*

1.6.1 Método de Escaneo Entrelazado

Este método presenta las siguientes características:

- Las líneas impares se despliegan primero, y posteriormente se despliegan las líneas pares, como se observa en la figura 1.22.

- Todas las líneas impares en conjunto reciben el nombre de *campo* (de manera similar, el conjunto de líneas pares se llama *campo*).

- El propósito original del escaneo entrelazado era el de evitar el “parpadeo” en pantalla.

- La televisión estándar utiliza este método, desplegando 60 campos por segundo (lo que da como resultado 30 cuadros por segundo) [5], [6].

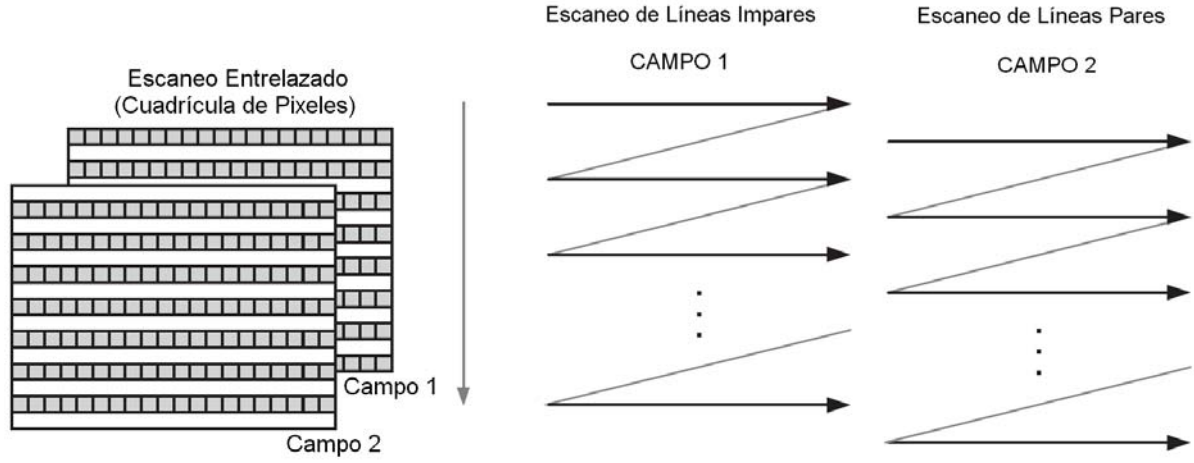


Figura 1.22 Método de escaneo entrelazado.

1.6.2 Método de Escaneo Progresivo

En este método, una imagen entera es escrita a un buffer, y posteriormente el buffer es desplegado completamente en pantalla. Para desplegar una imagen en pantalla utilizando este método, el despliegue se inicia desde la esquina superior izquierda de la pantalla, moviéndose hacia el lado derecho hasta completar una línea. Después de escanear una línea, se mueve a la línea de abajo para repetir el escaneo de izquierda a derecha. Este proceso se repite hasta que se completa un cuadro y la pantalla es refrescada. El escaneo progresivo es muy utilizado en monitores de computadora [5], [6].

En la figura 1.23 se muestra el trazado de pixeles por línea sobre la pantalla, usando el método de escaneo progresivo.

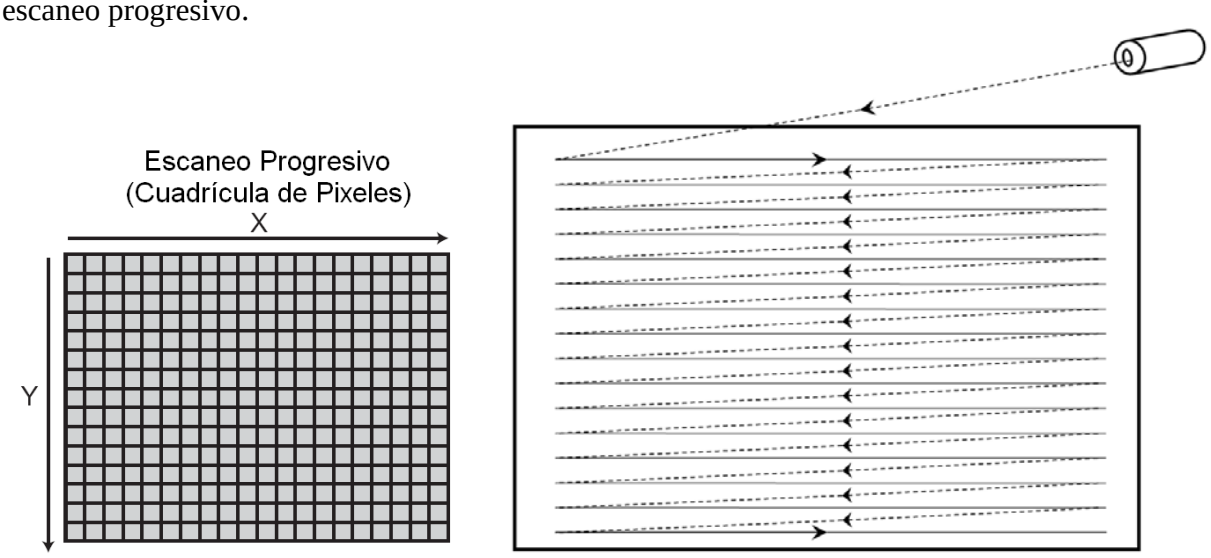


Figura 1.23 Método de escaneo progresivo y su trazado de pixeles en pantalla.

1.6.3 Pantalla de Tubo de Rayos Catódicos (CRT)

Un tipo de pantalla muy popular es el de *tubo de rayos catódicos (CRT)* en el cual la imagen se forma escaneando un haz modulado de electrones sobre una pantalla fosforescente. Un CRT es un tubo al vacío con forma de botella alargada. Un cañón de electrones en la parte trasera del tubo produce un haz de electrones el cual es dirigido hacia el frente del tubo. La superficie interior del frente del tubo está cubierta con una sustancia de fósforo la cual emite luz cuando es golpeada por los electrones. Para poder visualizar el color, los puntos de fósforo que producen el *rojo, verde y azul*, son colocados en el interior de la pantalla en patrones triangulares.

El color en una pantalla CRT se produce bombardeando a los tres diferentes puntos de fósforo en la pantalla (rojo, verde y azul) con los tres haces de electrones separados, los cuales son enfocados a los puntos de fósforo de cada color como se muestra en la figura 1.24.

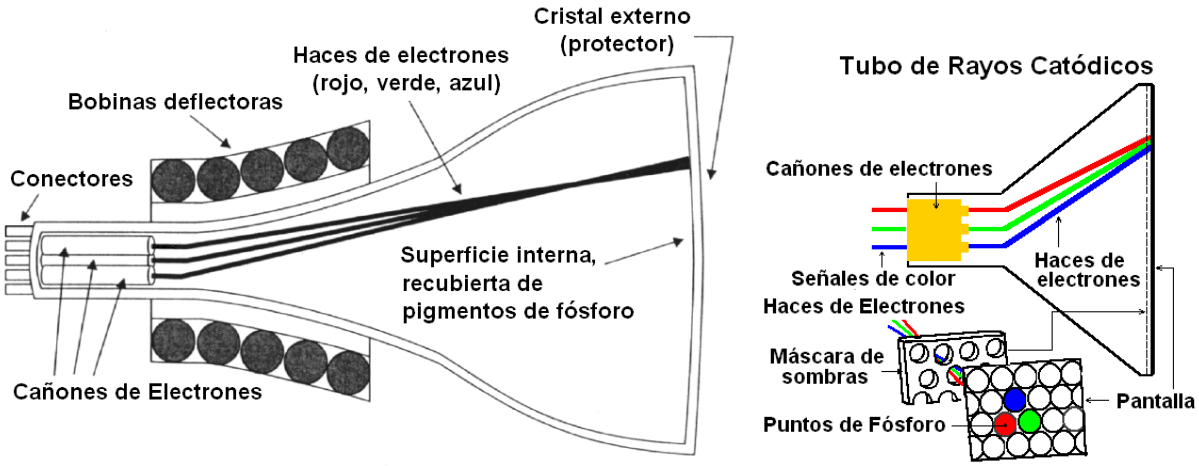


Figura 1.24 Esquema de una pantalla CRT.

Los puntos en la pantalla del CRT están muy cercanos entre si de tal manera que el ojo humano los percibe como un único punto. Variando la proporción de la intensidad de los tres haces se puede construir el triángulo de tres puntos que puede representar cualquier color deseado, incluyendo el blanco y negro. Si los tres haces están apagados, el color del punto resultante será el negro, y si los tres haces están a su máxima intensidad el color resultante de esta combinación será el blanco [9].

Las pantallas CRT utilizan el método de escaneo progresivo o el entrelazado, dependiendo del tipo de aplicación. Para evitar el efecto de "parpadeo" entre líneas, el fósforo utilizado en los CRT continúa emitiendo luz después de ser estimulado por el haz de electrones. El tiempo pertinente para que la intensidad de la luz de salida comience a disminuir se elige considerando que la emisión de luz debe conservarse hasta que inicie el siguiente ciclo de escritura [6].

La tecnología CRT es relativamente de bajo costo, sin embargo, tiene los inconvenientes de que un CRT requiere de una ruta lo suficientemente larga para el recorrido del haz de electrones, lo que lo hace un dispositivo muy voluminoso y además el tubo al vacío es muy pesado. Las *pantallas de cristal líquido (LCDs)* son otra alternativa al igual que las pantallas planas de plasma (su análisis a detalle esta fuera de este estudio).

RESUMEN

En este capítulo se han expuesto los fundamentos de las imágenes digitales, tanto vectoriales como de mapa de bits, siendo estas últimas las de nuestro interés. Además, se hizo un análisis de las principales características del sistema visual humano y su importancia en los sistemas de video digital.

Por otro lado, se estudiaron los diferentes espacios de color, analizando las ecuaciones para hacer la conversión de un espacio a otro. En este capítulo, también se describieron los dos tipos de sensores de video más populares en el mercado, los de tipo CCD y los CMOS, analizando sus ventajas y desventajas.

Además, se mencionaron las partes importantes del video digital, como la captura, formatos de muestreo, submuestreo y tasa de cuadros, así como formatos y estándares. Finalmente, se estudió la parte de visualización de video, haciendo énfasis en las pantallas CRT, las cuales son unas de las más populares.

La mayoría de los temas expuestos en éste capítulo, nos ayudaran a comprender adecuadamente el desarrollo y justificación del proyecto del trabajo de tesis.

2.- COMPRESION DE VIDEO DIGITAL

Dado que el video digital requiere del manejo de una gran cantidad de información, su uso no sería posible en ambientes de transmisión y almacenamiento, sino fuera por el uso de la compresión, el cual es un proceso que reduce el tamaño de la información requerida para la transmisión o almacenamiento del video digital.

El objetivo de este capítulo es describir los conceptos básicos de compresión de video digital, además, se analiza la codificación espacial, temporal y entrópica, el proceso de cuantización, así como algunas técnicas empleadas para realizar la compresión de video. Finalmente, se hace una descripción de los estándares y métodos de compresión de video más populares.

2.1 Conceptos de compresión de imágenes y video

La ***compresión*** es un proceso que permite que los datos que van a ser transmitidos o almacenados requieran menos bits que la información original, y que al aplicar el proceso de recuperación de los datos (***descompresión***) la información sea lo más parecida a la original. La ***compresión de video*** se refiere a las técnicas que reducen el número de bits requeridos para la transmisión o almacenamiento de las secuencias de imágenes de video [8].

A un dispositivo o programa que comprime una señal se le llama ***codificador*** y a un dispositivo o programa que descomprime una señal ***decodificador***. Cuando el dispositivo o programa incluye un codificador y decodificador recibe el nombre de ***CODEC*** [1].

En la figura 2.1 se muestra el diagrama a bloques de la compresión de datos de video en sistemas de transmisión y almacenamiento. Los algoritmos de compresión reducen el número total de parámetros necesarios para representar a la señal, estos parámetros son codificados en paquetes de datos para su transmisión o almacenamiento. Los algoritmos utilizados para comprimir imágenes y video producen, en menor o mayor grado, efectos o artefactos visibles indeseados, por lo que el video reconstruido debe tener una calidad razonable para el tipo de aplicación utilizada; así, la calidad que se requiere en una aplicación de video en directo por Internet es muy inferior a la que se exige para la transmisión de televisión. Además, la complejidad de los cálculos empleados debe ser viable para el tipo de aplicación, con un computo razonable y con un tiempo de procesamiento muy corto [6], [13].

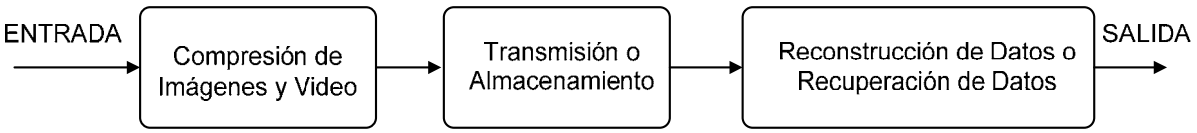


Figura 2.1 Compresión de imágenes y video, transmisión o almacenamiento, y descompresión.

La ***tasa de bits*** (también conocida como tasa de codificación) es un parámetro importante en la compresión de imágenes y video y frecuentemente es expresada en *bits por segundo (bits/seg, ó bps)*. Este término es apropiado para sistemas de transmisión. En aplicaciones de almacenamiento de imágenes, la tasa de bits frecuentemente es expresada en unidades de *bits por pixel (bpp)* [13].

Necesidad y beneficios de la compresión de video

El uso de video digital sin comprimir requiere del manejo de una gran cantidad de datos, por lo que su uso no sería posible en ambientes de transmisión y almacenamiento, siendo necesaria la compresión de video. La compresión de video presenta los siguientes beneficios [12], [13]:

- Optimiza el espacio de almacenamiento.
- Reduce el ancho de banda necesario para la transmisión.
- Reduce el tiempo de transmisión.
- Optimiza los recursos de transmisión y almacenamiento.

Los métodos de compresión pueden ser *sin pérdidas* o *con pérdidas* como se describe a continuación:

Compresión sin pérdidas: En este método la información recuperada es idéntica a la información original. Este tipo de compresión se utiliza en la codificación de datos binarios de aplicaciones informáticas en las que es absolutamente necesario recuperar la información original. En el caso de las imágenes, la compresión sin pérdidas encuentra su aplicación en la codificación de imágenes médicas o científicas en las que puede resultar crítica la pérdida de parte de la información [4], [13].

Compresión con pérdidas: En este método la información recuperada es diferente a la información original, pero las pérdidas son tolerables siempre y cuando la calidad de las señales decodificadas sea aceptable. La compresión con pérdidas es más común en la codificación de señales de vídeo y audio. En la codificación con pérdidas no es necesario codificar los componentes de información que no son sensibles por los sistemas de percepción humana.

La principal ventaja de la compresión con perdidas es que se logran factores de compresión muy superiores (alrededor de 25% del tamaño del archivo original) a los que se obtienen con los métodos sin pérdidas (alrededor del 90%). Además, suelen ser métodos escalables con la aplicación, es decir, el grado de pérdida de calidad que se tolera, se puede adecuar al tipo de aplicación donde se utilice el codificador [1], [14].

Tipos de redundancia en las señales de video

Las señales de imágenes y video tienen propiedades que pueden ser explotadas por los codificadores. El análisis estadístico de las señales de video indica que hay una fuerte correlación entre los cuadros de imágenes sucesivas y dentro de los mismos elementos de la imagen. Los tipos de redundancia presentes en las señales de video son:

- ***Redundancia espacial:*** Esta se presenta cuando los pixeles contiguos dentro de una imagen o un cuadro de video tienden a estar altamente correlacionados.

- ***Redundancia temporal:*** Esta se da cuando las regiones contiguas entre los cuadros de imágenes sucesivas tienden a estar altamente correlacionados.

En teoría, la decorrelación de estas señales puede llevar a una compresión de la información sin afectar la resolución de la imagen.

- ***Redundancia entre símbolos:*** Un codificador puede reducir la redundancia entre los símbolos de los datos comprimidos, utilizando técnicas de codificación de longitud variable (*codificación entrópica*).

- ***Redundancia subjetiva o perceptiva:*** Por otro lado, se tiene la llamada *redundancia subjetiva o perceptiva*, en donde se explotan las limitaciones del sistema visual humano para percibir cierta información visual espacio-temporal, para lograr una mayor reducción. Por ejemplo, el sistema visual humano es mucho más sensible a bajas frecuencias que a altas frecuencias por lo que es posible comprimir una imagen eliminando ciertos componentes de alta frecuencia *(sección 1.2)*.

2.2 Codificación Espacial o Intraframe

La compresión de una imagen de video en el dominio espacial es llamada ***Codificación Espacial o Intraframe*** y es la base para la codificación *JPEG (Unión Grupal de Expertos en Imágenes)*. La *codificación espacial* disminuye la redundancia espacial entre los pixeles pertenecientes a una imagen, empleando algún método de compresión de datos, como por ejemplo la codificación con transformadas.

En la compresión espacial, cada pixel puede ser cuantizado por separado, pero esto puede ocasionar efectos indeseables en algunas imágenes. Una alternativa es la cuantización de vectores, donde un grupo de pixeles se cuantizan juntos. Esto teóricamente ofrece una compresión más eficiente, pero en la práctica requiere cálculos computacionales muy intensos y un tiempo de procesamiento muy grande. Una alternativa es la codificación por transformadas, donde un bloque de pixeles se transforma al dominio de la frecuencia y después se cuantiza. Posteriormente se puede utilizar la codificación entrópica para reducir aún más los datos [6], [11].

2.2.1 Modulación por Codificación Diferencial de Pulso (DPCM)

Los métodos predictivos intentan reducir la redundancia espacio/temporal de una señal haciendo una estimación de cada muestra en función de las muestras anteriores. La diferencia entre una muestra y su predicción, si esta última está bien calculada, se convierte en una secuencia con un grado de correlación muy pequeño, lo que equivale a decir que la redundancia se ha eliminado.

La técnica de ***codificación diferencial*** codifica las diferencias entre la misma señal y su predicción. Por lo tanto, ésta también se conoce como ***codificación predictiva***. La codificación diferencial se basa principalmente en dos componentes: predicción y cuantización. Cuando la diferencia de la señal es cuantizada, la codificación diferencial es llamada ***DPCM*** [13].

Codificación diferencial sin pérdidas

El esquema básico de un codificador diferencial sin pérdidas se muestra en la figura 2.2. En el caso de imágenes y video, la señal de entrada puede ser una línea de luminancia de la imagen. La codificación diferencial consiste en calcular la diferencia entre dos muestras consecutivas. Esta transformación es invertible, por lo que la información original se puede recuperar de forma exacta a partir de los datos codificados [4].

Analíticamente, la codificación diferencial viene dada por la ecuación (2.1), mientras que la codificación diferencial inversa se obtiene por la ecuación (2.2) [4]:

$$y[n] = x[n]-x[n-1] \tag{2.1}$$

donde: *x[n]* es la señal de entrada y *y[n]* es la señal codificada.

$$z[n] = y[n]+z[n-1] \tag{2.2}$$

donde: *z[n]* es la señal recuperada y *y[n]* es la señal codificada.

Para verificar que un proceso es el inverso del otro, se pueden aplicar las ecuaciones (2.1) y (2.2) a las tablas de valores que se muestran en la figura 2.2 [4].

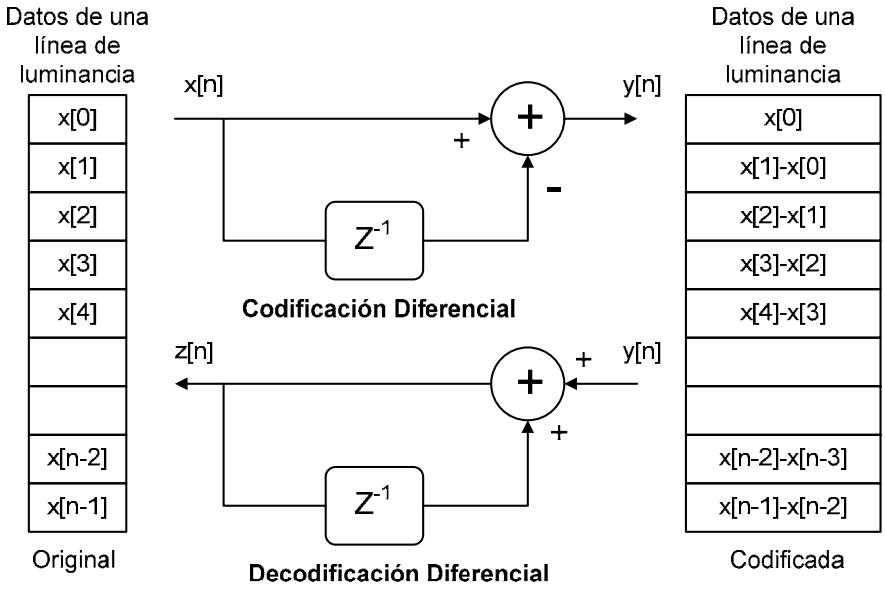


Figura 2.2 Codificación y decodificación diferencial.

La codificación diferencial se puede considerar un caso particular de la *predicción de muestras*. Los *predictores lineales* son uno de los métodos más utilizados para la codificación de señales de audio y video y se basan en estimar el valor de la muestra actual a partir de una combinación lineal de las muestras anteriores. El método general de un predictor se ilustra en la figura 2.3, donde se indican tanto la codificación como la decodificación. Si la predicción de la muestra es correcta, la señal que se codifica es el *error de predicción*, el cual tendrá un valor cercano a cero, por lo que podrá codificarse de forma eficiente.

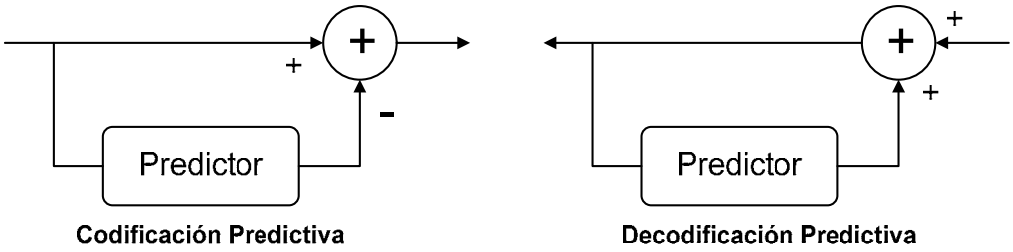


Figura 2.3 Esquema general de codificación por predicción de muestras.

La ecuación (2.3) describe el funcionamiento del predictor [4]:

$$\tilde{x}[n] = \sum_{k=1}^{p} a_k \bullet x[n-k] \tag{2.3}$$

donde:

- ***p*** es el *orden del predictor,* el cual es el número de muestras anteriores utilizadas para realizar la predicción de la muestra actual.
- $\boldsymbol{a_k}$ son los *coeficientes de predicción*, los cuales pueden calcularse con base a las características estadísticas de la señal [4].

En la codificación de imágenes, la predicción más simple se realiza tomando el valor del pixel anterior, como se ilustra en el pixel A de la figura 2.4. Este tipo de codificación se puede representar con la ecuación (2.1) y se considera un predictor de orden 1, con un coeficiente de predicción igual a 1 según la ecuación (2.3).

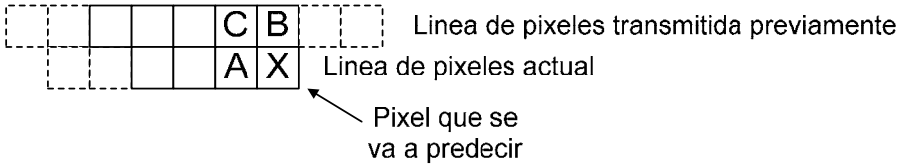


Figura 2.4 Codificación DPCM.

Para obtener una predicción más exacta se puede utilizar un promedio ponderado de los pixeles contiguos (por ejemplo, A, B y C en la figura 2.4). Al valor actual del pixel X se le resta el valor de la predicción, y la diferencia (el error de predicción) es la que se transmite al receptor. El error de predicción típicamente es pequeño debido a la alta correlación espacial, así que para efectuar la compresión, los errores de predicción más pequeños, los más comunes, se representan con códigos binarios cortos y los errores de predicción más grandes, los menos comunes, se representan con códigos más largos [1]. Esta es la idea básica de la codificación entrópica, la cual examinaremos más adelante.

En la figura 2.5 se muestran los resultados al aplicar el codificador diferencial a una línea de una imagen, donde se grafican el nivel de gris de la línea original de la imagen y los valores resultantes de la codificación. En la figura se observa que los valores codificados se concentran alrededor de cero y que únicamente en los contornos de la imagen se producen niveles significativos en la señal diferencial [4].

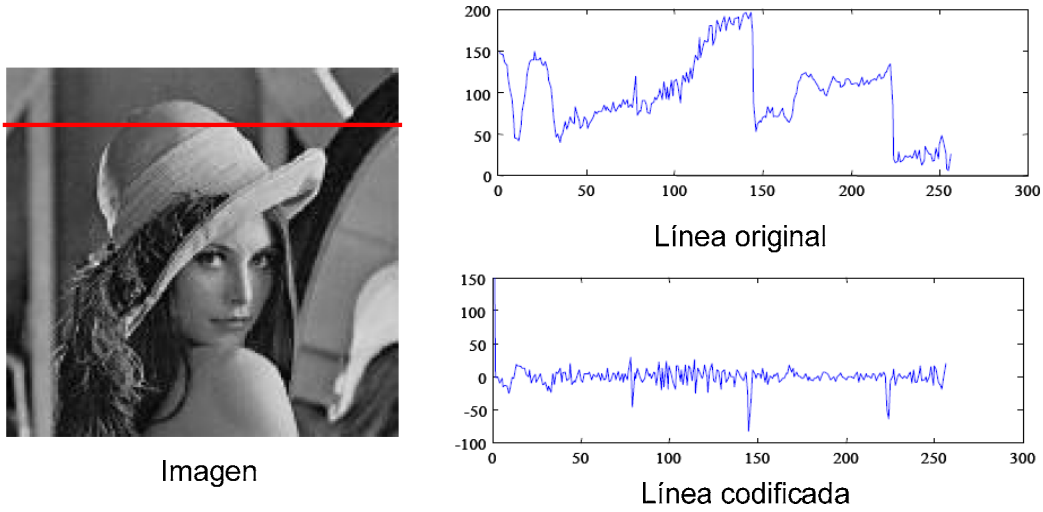


Figura 2.5 Niveles de gris de la línea de una imagen antes y después de la codificación diferencial.

La figura 2.6 muestra la imagen de diferencias que se obtiene al aplicar el codificador diferencial a cada una de las líneas de la imagen en blanco y negro de la figura 2.5. La imagen original está representada con 256 niveles de gris (8 bits por pixel), donde el nivel 0 corresponde al color negro y el 255 al blanco. Para efectos de visualización, la imagen de diferencias se normaliza tomando el nivel 0 normalizado como un gris neutro, lo que ocasiona que los valores negativos sean más oscuros que el gris neutro y los valores positivos más claros. En la imagen de diferencias la mayoría de los pixeles están próximos al gris neutro (nivel 0 normalizado) y sólo en los contornos de la imagen aparecen pixeles cuyo valor difiere notablemente de cero [4].

Figura 2.6 Imagen Original e imagen de diferencias obtenida por codificación diferencial.

El hecho de que la mayoría de los pixeles estén muy cercanos a cero (gris neutro) se aprecia con más claridad mediante la representación con histogramas. Un histograma es una representación bidimensional de la frecuencia relativa con la que aparece cada nivel de gris en la imagen. Así, para cada nivel de gris posible (eje de las abscisas), encontramos el número de pixeles en la imagen con ese nivel de gris. Por lo tanto, el ***histograma*** *es una representación aproximada de la probabilidad de que un pixel tome un determinado nivel de gris en la imagen* [4].

Las gráficas de la figura 2.7 representan los histogramas de la imagen original y de la imagen de diferencias de la figura 2.6. Nótese que en éstas gráficas los niveles de gris se han normalizado, de tal forma que el negro corresponde al cero mientras que el blanco corresponde a la unidad. En el histograma de la izquierda, que corresponde a la imagen original, los niveles de gris se distribuyen dentro de toda la escala sin concentrarse directamente en algún valor. En cambio, en la gráfica de la derecha, correspondiente a la imagen de diferencias, existe una gran concentración de elementos de la imagen con un nivel de gris próximo a 0.5 (el gris neutro con el que hemos representado el valor cero en la imagen de diferencias).

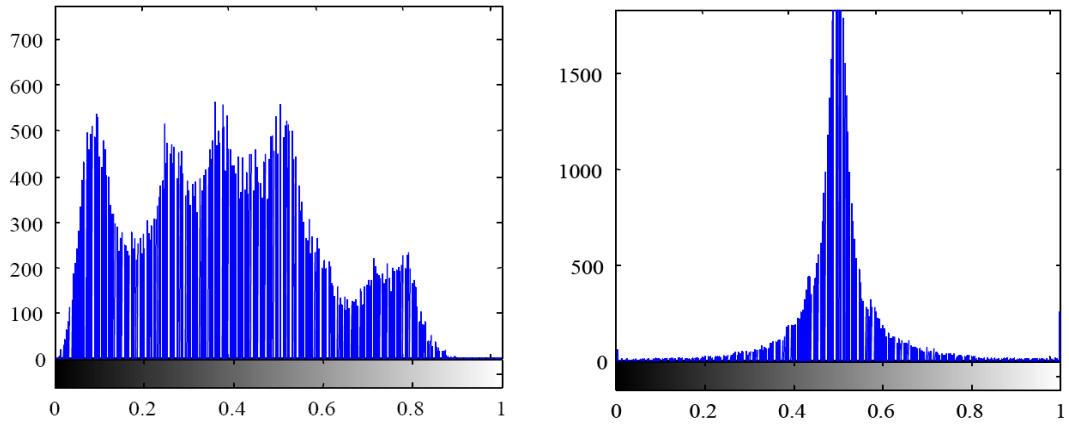


Figura 2.7 Histogramas de la imagen original y de la imagen de diferencias.

Codificación DPCM

La codificación diferencial se puede efectuar mediante la cuantización y reducción de precisión del error de predicción; ésta codificación es llamada ***DPCM*** y es una codificación con pérdidas, ya que se vuelve imposible recuperar los valores originales al aplicar la decodificación.

La mejor predicción se realiza a través de los pixeles vecinos, ya sean pixeles del mismo cuadro, de un cuadro previo, o una combinación de ambos. La codificación DPCM puede ser aplicada espacialmente, utilizando los pixeles adyacentes en el mismo cuadro *(codificación predictiva Intraframe)* o temporalmente, usando los pixeles adyacentes en un cuadros previo para formar la predicción *(codificación predictiva Interframe),* obteniendo una compresión pequeña con baja complejidad. La combinación de ambas es llamada *codificación predictiva* [1], [11].

2.3 Codificación mediante la Transformada Coseno Discreta (DCT)

La transformación de los datos es un procedimiento muy utilizado en los codecs actuales, y se utiliza para representar la información en una forma alternativa, resultando más evidente la redundancia existente en los datos originales. Es necesario que esta transformación sea invertible, es decir, que a partir de los datos transformados se pueda recuperar la información con el mayor parecido posible a la información original [4].

En la codificación por transformadas, las muestras de la imagen se transforman a otro dominio (o representación) y se representan por los *coeficientes de la transformada*. En el dominio espacial (la forma original de la imagen), los pixeles tienen una alta correlación espacial. El objetivo de la codificación por transformadas es reducir esta correlación, dejando idealmente un número pequeño de coeficientes significativos (los más importante para la apariencia de la imagen original) y un gran número de coeficientes no significativos (éstos pueden ser descartados sin afectar significativamente la calidad visual de la imagen). El proceso de transformación no realiza la compresión por si solo: por lo general le sigue un proceso de cuantización con pérdidas en el cual se remueven los coeficientes no significativos, dejando solo un pequeño número de coeficientes significativos. Este proceso se ilustra en la figura 2.8. La codificación por transformadas es la base de muchos de los sistemas de compresión de imágenes y video [1].

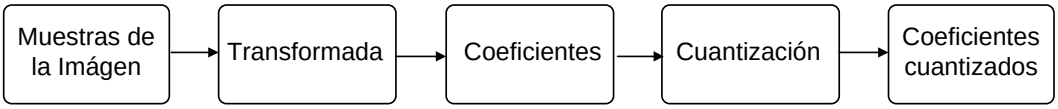


Figura 2.8 Codificación por transformada.

Aunque se han propuesto diferentes transformadas, en la actualidad la transformada coseno discreta es la más utilizada por los codecs de video. Existen otras transformadas que se han desarrollado para la codificación de video, como la *Transformada Discreta de Fourier (DFT),* la *Transformada Karhunen Lo'eve (KLT)* y la *transformada Wavelet*. La DFT es poco utilizada debido a que sufre discontinuidades en los límites de bloque. Para imágenes del mundo real, la DCT supera a la DFT en la compresión de energía. La KLT presenta la descomposición más óptima, pero requiere un gran número de operaciones, por lo que necesita mayor tiempo de procesamiento [6].

La transformada Wavelet es una alternativa a la transformada coseno discreta, ya que mejora los factores de compresión y no sufre de los efectos de bloque, los cuales muchas veces son visibles en la compresión basada en la DCT. Esta transformada es utilizada en codecs de compresión de imágenes, como *JPEG2000* y en codecs de compresión de video como *MPEG-4 (Grupo de Expertos de Imágenes en Movimiento - 4)* [4], [6]. En este trabajo solo se analiza la DCT, la cual es utilizada por el estándar JPEG. El análisis de las otras transformadas esta fuera de este estudio.

La Transformada Coseno Discreta

En 1974 Ahmed Natarajan y Rao propusieron la DCT. Desde entonces, ésta ha sido la transformada más popular para la codificación de imágenes y video, y es una parte fundamental en la compresión de imágenes JPEG. Principalmente, existen dos razones para su popularidad: la primera, es muy efectiva en la transformación de los datos de la imagen en un formato fácil de comprimir, y la segunda, la DCT puede ser implementada eficientemente tanto en software como en hardware [1].

La DCT presenta las siguientes propiedades, las cuales son muy útiles para la compresión de imágenes y video [4]:

1) ***Compactación de la energía en el dominio transformado:*** La transformada coseno discreta concentra la mayor parte de la información en unos pocos coeficientes transformados. Esto permite obtener una codificación eficiente de la imagen, puesto que basta con codificar de forma precisa los coeficientes transformados para obtener una buena representación de todo el bloque de la imagen. Se debe tener en cuenta que la compactación de la energía en unos pocos coeficientes es un parámetro puramente estadístico, lo que significa que siempre es posible encontrar un bloque de una imagen en el que la energía en el dominio transformado esté dispersa entre todos los coeficientes. No obstante, esto tiene una probabilidad de ocurrencia muy baja y no suele producirse si trabajamos con imágenes naturales.

2) ***Algoritmos eficientes para el cálculo rápido de la DCT:*** Existen algoritmos rápidos para realizar la DCT. La restricción para poder utilizar estos algoritmos es que los bloques tengan un tamaño que sea múltiplo de una potencia de dos. Si no es así, se deben añadir ceros a los bloques, aunque se aumenta el riesgo de aparición del efecto de bloques al decodificar la imagen.

3) ***Errores reducidos en los contornos de los bloques.*** Para evitar la aparición del efecto de bloques indeseados en la imagen reconstruida, es necesario que los errores de codificación en los límites de los bloques sean muy pequeños.

La DCT frecuentemente es referida como *Transformada Coseno Discreta Directa (FDCT)*. La FDCT transforma un conjunto de muestras de la imagen (en el dominio espacial) en un conjunto de coeficientes de funciones coseno con frecuencias crecientes (dominio de la transformada). Cada uno de los valores originales es transformado en una suma de cosenos.

La DCT es una transformada invertible, lo que significa que sus coeficientes de salida pueden ser utilizados para reconstruir los valores originales de entrada, el inverso de la DCT es llamado *Transformada Coseno Discreta Inversa (IDCT)*. La IDCT reconstruye un bloque de muestras de la imagen a partir de un bloque de coeficientes DCT.

La DCT se utiliza comúnmente para procesar datos en una o dos dimensiones. El número de valores de entrada regularmente es una potencia de dos. En JPEG la DCT y la IDCT siempre se utilizan en 2 dimensiones aplicadas sobre datos ordenados en bloques de 8x8. La versión unidimensional (1-D) transforma las muestras de un vector de una dimensión en un vector unidimensional de coeficientes, mientras que la versión bidimensional (2-D) transforma un bloque de muestras en un bloque de coeficientes. En la figura 2.9 se muestran las dos formas de la DCT, la forma directa en una dimensión (FDCT 1-D), y la forma directa en dos dimensiones FDCT-2D [1], [3].

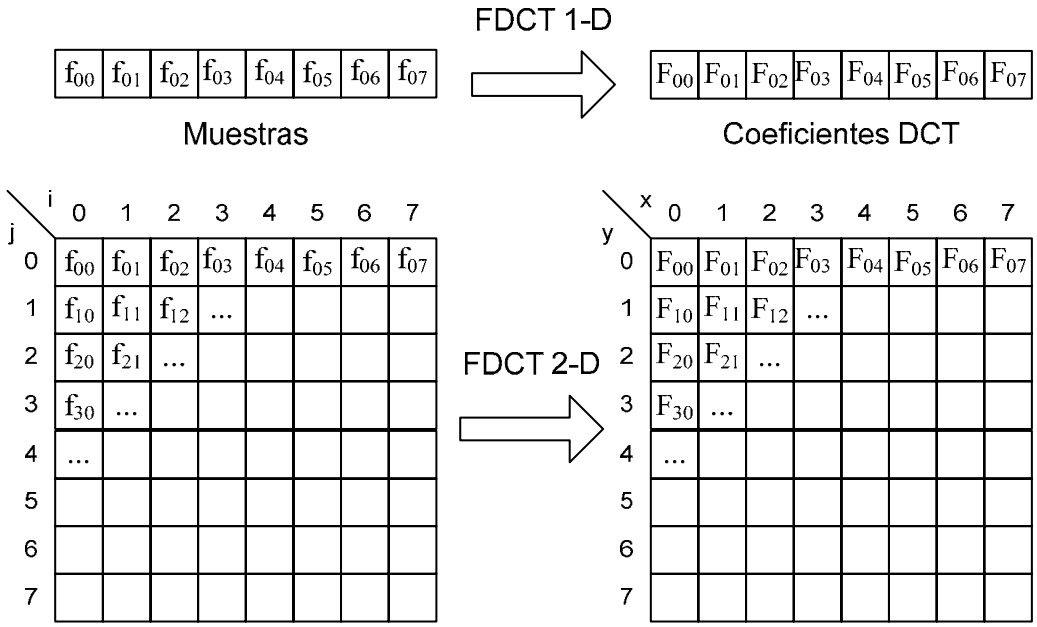


Figura 2.9 Transformada Coseno Discreta en 1-D y 2-D.

La DCT en dos dimensiones es un proceso separable que se implementa utilizando dos DCT's unidimensionales: una en la dirección horizontal seguido por otra en la dirección vertical. Para un bloque de *M x N* pixeles, la DCT unidimensional para *N* pixeles está dada por la ecuación (2.4) [11]:

$$F(u) = \sqrt{\frac{2}{N}} C(u) \sum_{x=0}^{N-1} f(x) \cos\left[\frac{(2x+1)u\pi}{2N}\right] \quad u = 0, 1, \ldots, N\text{-}1 \tag{2.4}$$

donde:

$C(u) = \frac{1}{\sqrt{2}}$ para u, v = 0

$C(u)$ = 1 para u > 0

$f(x)$ representa la intensidad del pixel x

$F(u)$ representa los N coeficientes de la transformada

La IDCT unidimensional está definida por la ecuación (2.5) [11]:

$$f(x) = \sqrt{\frac{2}{N}} \sum_{x=0}^{N-1} C(u) F(u) \cos\left[\frac{(2x+1)u\pi}{2N}\right] \quad x = 0, 1, \ldots, N\text{-}1 \tag{2.5}$$

Nótese que el factor de normalización $\sqrt{1/N}$ se utiliza para que la transformación sea ortonormal, es decir, para que la energía tanto en el dominio espacial como en el dominio de la transformada sea igual. En los codecs estandarizados, el factor de normalización para la DCT bidimensional está definido como ½. Esto deja a los coeficientes en el intervalo de -2047 a +2047 [11].

La transformada coseno discreta puede extenderse de forma directa a dos dimensiones. Las ecuaciones (2.6) y (2.7) definen a la FDCT y a la IDCT en 2-D para un bloque de 8x8 [1], [4]:

DCT:

$$F(u,v) = \frac{C(u)}{2}\frac{C(v)}{2}\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right] \qquad (2.6)$$

$$u, v \rightarrow 0:7$$

IDCT:

$$f(x,y) = \sum_{u=0}^{7}\sum_{v=0}^{7}\frac{C(u)}{2}\frac{C(v)}{2}F(u,v)\cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right] \qquad (2.7)$$

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

donde:

f(x,y) es el valor de la posición del pixel en la posición *(x,y)* dentro del bloque

F(u,v) es el valor del coeficiente DCT *(u,v)*

C(u), C(v) = $\frac{1}{\sqrt{2}}$ para u, v = 0

C(u), C(v) = 1 para u, v > 0

La DCT representa cada bloque de las muestras de la imagen como una suma ponderada de las funciones coseno en 2-D (*funciones base*). Las funciones están graficadas como funciones base de 8x8 pixeles en la figura 2.10. La función base de la esquina superior izquierda tiene la frecuencia más baja y es solo un bloque uniforme. Este coeficiente es conocido como el coeficiente DC y los otros son llamados coeficientes AC. El coeficiente DC proporciona el promedio de todo el bloque de muestras [1], [19].

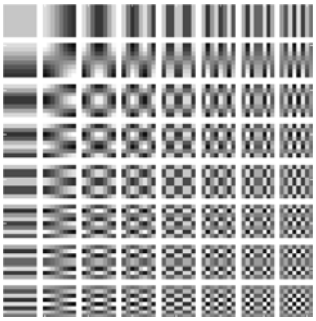


Figura 2.10 Funciones base de la DCT para N = 8.

Moviéndose a la derecha en la figura 2.10, las funciones base contienen un número creciente de *ciclos* entre zonas oscuras y luminosas en la dirección horizontal: esto representa el incremento de la frecuencia espacial horizontal. Moviéndose hacia abajo las funciones contienen un incremento en la frecuencia espacial vertical. Moviéndose diagonalmente a la derecha y abajo, las funciones contienen incrementos tanto en las frecuencias horizontales y verticales. El bloque de muestras puede ser reconstruido mediante la suma de las 64 funciones base, cada una multiplicada por un factor, el cual es el coeficiente DCT correspondiente a F(u,v) [1].

2.4 Algoritmos rápidos y optimizados para el cálculo de la DCT y la IDCT

Para calcular los coeficientes de la transformada, cada transformación requiere de *7 sumas y 8 multiplicaciones* utilizando la DCT o IDCT unidimensional. Este proceso es repetido para 64 coeficientes, tanto en la dirección horizontal como en la vertical. Dado que la compresión de video por software es muy demandada, los métodos para reducir la enorme carga computacional son altamente deseables.

De acuerdo a la ecuación (2.6), cada uno de los 64 coeficientes en la DCT es una función ponderada de todas las 64 muestras, lo que significa que se deben realizar *64 cálculos* para cada coeficiente de la DCT, donde cada cálculo es una operación de multiplicación-acumulación. En total se requieren de *64x64 = 4096 operaciones de multiplicación-acumulación* para ejecutar una DCT completa en un bloque de 8 x 8 muestras [1].

2.4.1 Transformadas separables

La transformada bidimensional puede obtenerse a partir de las transformadas unidimensionales, como se muestra en la figura 2.11. En primer lugar, se aplica la transformada unidimensional a cada una de las filas del bloque de la imagen original, con lo que se obtiene una imagen intermedia. Posteriormente, sobre los resultados obtenidos, se aplica nuevamente la transformada unidimensional, pero ahora sobre las columnas. Evidentemente, se pueden procesar primero las columnas de la imagen original, y después las filas de la imagen intermedia [4].

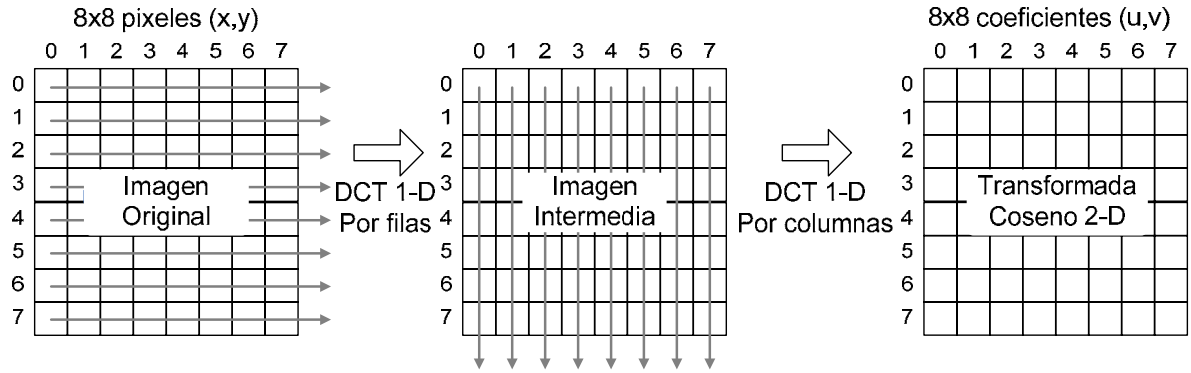


Figura 2.11 Cálculo de la DCT bidimensional a partir de transformadas unidimensionales.

La DCT bidimensional puede ser calculada aplicando repetidamente la DCT unidimensional. La DCT unidimensional para 8 muestras está dada por la ecuación (2.8) [1]:

$$F(u) = \frac{1}{2} C(u) \sum_{x=0}^{N-1} f(x) \cos\left[\frac{(2x+1)u\pi}{16}\right] \qquad u \to 0:7 \qquad (2.8)$$

donde:

$C(u) = \frac{1}{\sqrt{2}}$ para u, v = 0

$C(u) = 1$ para u > 0

$f(x)$ representa las muestras de entrada x (pixeles)

$F(u)$ representa los 8 coeficientes de la transformada

La IDCT también puede ser calculada utilizando dos IDCT's unidimensionales, de forma similar a la DCT. La ecuación para la DCT en 1-D para 8 muestras está dada por la ecuación (2.9) [1]:

$$f(x) = \sum_{x=0}^{N-1} \frac{1}{2} C(u) F(u) \cos\left[\frac{(2x+1)u\pi}{16}\right] \qquad (2.9)$$
$$x \rightarrow 0:7$$

El método para calcular la DCT bidimensional utilizando dos DCT unidimensionales en forma separada presenta las siguientes ventajas [1]:

- *Reduce el número de operaciones:* Para obtener los coeficientes de la imagen o matriz intermedia para un bloque de 8x8 utilizando la DCT unidimensional se requieren *8x64 = 512 operaciones de multiplicación-acumulación*. Para obtener los coeficientes de la DCT en dos dimensiones, *el número total de operaciones requerido es 2x8x64=1024*, lo cual reduce el número de operaciones al 25% con respecto al número de operaciones requerido al utilizar la ecuación (2.6), en donde el cálculo de la DCT bidimensional se realiza en forma directa [1].

- La DCT unidimensional puede manipularse fácilmente para reducir el número de operaciones o bien para simplificar los cálculos [1].

Las implementaciones prácticas de la FDCT y la IDCT caen principalmente en dos categorías [1]:

1) *Computación mínima:* Las operaciones de la DCT (1-D o 2-D) son reorganizadas para explotar la simetría inherente de las operaciones coseno para minimizar el número de multiplicaciones y/o sumas. Este enfoque es apropiado para implementaciones por software.

2) *Máxima regularidad:* Los cálculos para la DCT en 1-D o 2-D son organizados para regular el flujo de datos y el orden de procesamiento. El enfoque de regulación del flujo de datos es adecuado para implementaciones de hardware dedicado.

2.4.2 Cálculo de la DCT basado en los algoritmos de Wang, Suehiro y Hatori.

Como se ha mencionado, la DCT y la IDCT bidimensionales pueden calcularse a partir de la aplicación de dos DCTs unidimensionales. Existe un algoritmo muy eficiente para el cálculo de la DCT y la IDCT unidimensionales, el cual esta basado en los algoritmos desarrollados por Wang por un lado, y por Suehiro y Hatori por el otro. Este algoritmo optimizado para la DCT se muestra en la figura 2.12. En la figura, las flechas indican la dirección en que fluye la información, en caso de que las flechas estén modificadas por un factor la información se multiplica por este factor, y por último, cuando dos cantidades (flechas) se encuentran en un punto éstas se suman [20].

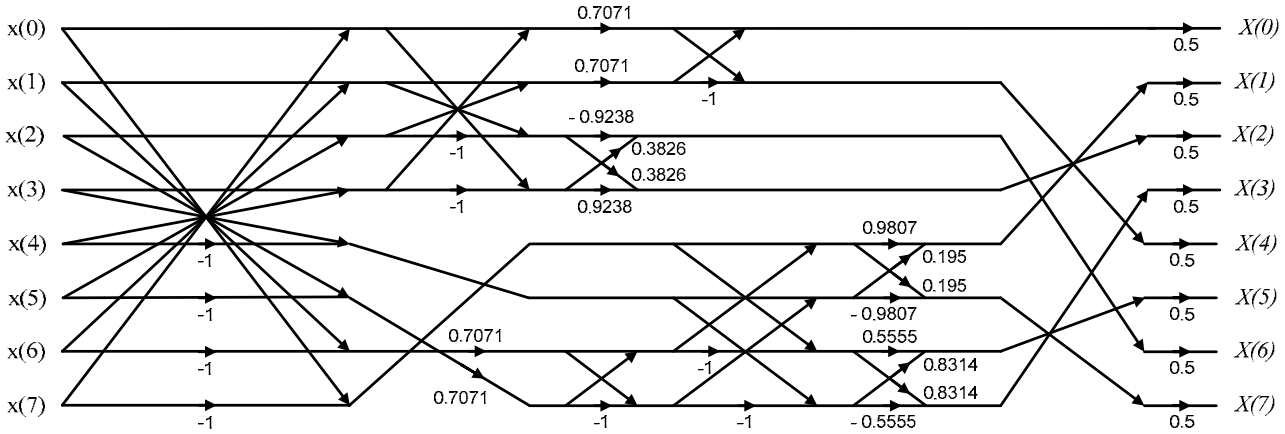


Figura 2.12 Algoritmo rápido y optimizado de la DCT.

En la figura 2.13 se muestra el algoritmo optimizado para el cálculo de la IDCT en 1-D [20]

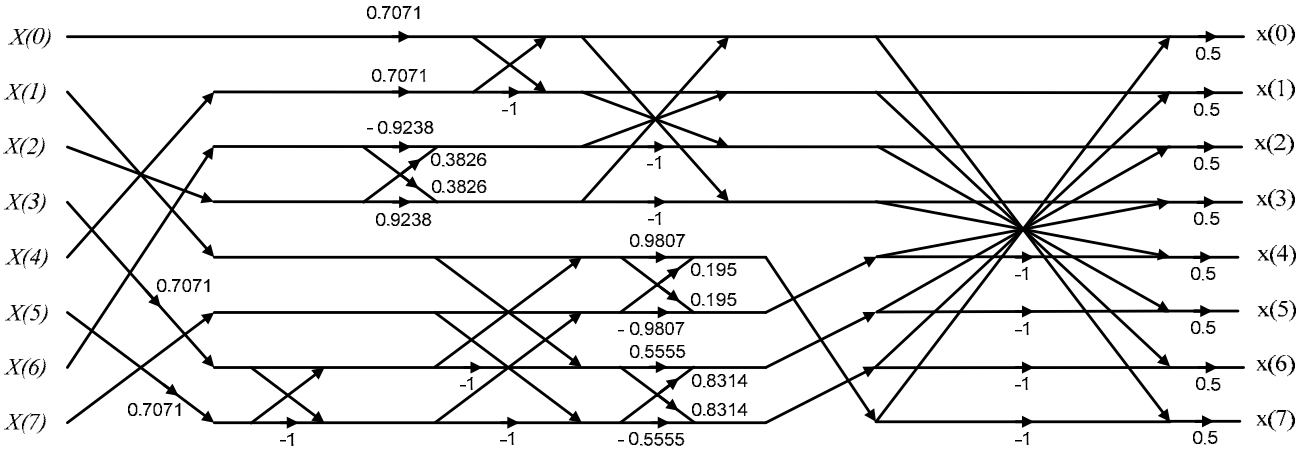


Figura 2.13 Algoritmo rápido y optimizado de la IDCT.

- Para el cálculo de la DCT unidimensional optimizada (basada en los algoritmos de Wang, y de Suehiro y Hatori) se requieren ***26 sumas y 16 multiplicaciones***, para un conjunto de 8 muestras.

- Para el cálculo de la IDCT unidimensional optimizada (basada en los algoritmos de Wang, y de Suehiro y Hatori) se necesitan ***27 sumas y 16 multiplicaciones***, para un conjunto de 8 muestras.

- Utilizando el criterio descrito en está sección para el cálculo de la DCT bidimensional, a partir de dos DCTs unidimensionales, se necesitan un total de ***416 sumas y 256 multiplicaciones***, para obtener los coeficientes DCT de un bloque de 8x8 muestras.

- De manera similar, para obtener la IDCT bidimensional se necesitan ***432 sumas y 256 multiplicaciones***, para un bloque de 8x8 muestras.

En la tabla 2.1 se hace una comparación del número de operaciones requeridas por el algoritmo convencional de la DCT y la IDCT unidimensionales, y el algoritmo basado en los desarrollos de Wang, y de Suehiro y Hatori, para un bloque de 8x8 muestras.

	DCT			***IDCT***		
	Sumas	***Multiplicaciones***	**Total**	***Sumas***	***Multiplicaciones***	**Total**
Algoritmo convencional	512	512	1024	512	512	1024
Algoritmo Wang, Suehiro y Hatori	416	256	672	432	256	688

Tabla 2.1 Comparación del número de operaciones para el cálculo de la DCT y la IDCT.

De la tabla 2.1 podemos observar que el número de operaciones en el algoritmo optimizado (basado en Wang, Suehiro y Hatori), *se reduce en un 65.63% para la DCT*, con respecto al algoritmo convencional, *y en un 67.19% para la IDCT*. Debido a la eficiencia del algoritmo optimizado de Wang, Suehiro y Hatori, se decidió utilizarlo en este trabajo.

2.5 Proceso de Cuantización

La ***cuantización*** es el proceso de conversión de una señal con un intervalo de X valores a una señal con un intervalo reducido de Y valores. Esto hace posible que la señal cuantizada se pueda representar con menos bits que la original, dado que el intervalo de valores posibles es más pequeño. Un ***cuantizador escalar*** asigna a una muestra de la señal de entrada un valor de salida cuantizado y un ***cuantizador vectorial*** asigna a un grupo de muestras de entrada (un vector) un grupo de valores cuantizados [12].

En la compresión de imágenes y video, el proceso de codificación por transformadas no realiza la compresión por si mismo. La transformación ocasiona que las partes de energía mas significativas de la imagen se concentren en los componentes de baja frecuencia, provocando que la mayoría de los coeficientes tengan poca energía. La cuantización y la codificación de longitud variable de los coeficientes de la transformada son los que se encargan de reducir la tasa de bits. El propósito de la cuantización es remover los componentes de los datos transformados que no son importantes en la apariencia visual de la imagen, y retener los componentes visuales importantes. Una vez removidos, los componentes menos importantes no pueden ser recuperados, por lo que la cuantización es un proceso con pérdidas.

Por otra parte, ya que el ojo humano es menos sensible a las distorsiones en altas frecuencias, a estos coeficientes se les puede aplicar una cuantización más amplia, para lograr una mayor compresión. La cuantización con un tamaño de escalón QF o factor de paso mas amplio ocasiona que un mayor número de coeficientes se hagan cero y por lo tanto se obtenga una mayor compresión, pero con perdidas en la calidad de la imagen [1], [11].

2.5.1 Cuantización Escalar

En los codecs de compresión de imágenes y video, la operación de cuantización normalmente se realiza en dos partes: un *cuantizador directo* en el codificador y un *cuantizador inverso* en el decodificador [12].

Un parámetro crítico en la cuantización escalar es el *tamaño de escalón QP* entre los valores sucesivos cuantizados. Si el tamaño de escalón es grande, el intervalo de valores cuantizados será pequeño, por lo que puede ser representado eficientemente durante la transmisión (compresión alta). Sin embargo, los valores cuantizados serán una aproximación muy pobre de la señal original. Si el tamaño de escalón es pequeño, los valores cuantizados serán mucho mas parecidos a la señal original, pero se tendrá un intervalo de valores cuantizados más grande, lo que reduce la eficiencia de la compresión [12].

Un ejemplo sencillo de cuantización escalar es el proceso de redondeo de un número fraccionario al entero más cercano. Este es un proceso con perdidas, ya que no es posible determinar el valor exacto del número fraccionario original a partir del valor redondeado. En la ecuación (2.10) se muestra un ejemplo más general de un cuantizador escalar [12]:

$$S_q = round\left(\frac{X}{QP}\right) \tag{2.10}$$

donde: ***X*** es el *valor de la muestra original.*
S_q es el *valor de la muestra cuantizada*.
QP es el *tamaño de escalón de cuantización*.

La ecuación (2.11) muestra el proceso inverso de cuantización de la ecuación (2.10).

$$Y = S_q \bullet QP \tag{2.11}$$

donde:
Y es el *valor de la muestra después de aplicar el proceso inverso de cuantización.*

En la tabla 2.2 se muestra un ejemplo con los valores de las muestras originales *X*, y los valores de las muestras recuperadas *Y* después de aplicarles el proceso inverso de cuantización, utilizando diferentes tamaños de escalón QP. Los valores de *Y* se calculan con la ecuación Y=*QP●round(X/QP)*, la cual se obtiene sustituyendo la ecuación (2.10) en la ecuación (2.11) [12].

	Y			
X	***QP = 1***	***QP = 2***	***QP = 3***	***QP = 5***
-4	-4	-4	-3	-5
-3	-3	-2	-3	-5
-2	-2	-2	-3	0
-1	-1	0	0	0
0	0	0	0	0
1	1	0	0	0
2	2	2	3	0
3	3	2	3	5
4	4	4	3	5
5	5	4	6	5
6	6	6	6	5
7	7	6	6	5
8	8	8	9	10
...				

Tabla 2.2 Ejemplo del proceso inverso de cuantización escalar.

Como se observa en la tabla 2.2, mientras más grande sea el tamaño de escalón, mayor será el número de muestras que se hagan cero, aunque esto puede repercutir en la calidad de las imágenes y el video recuperado.

2.6 Codificación Temporal o Interframe

Las secuencias de imágenes de video regularmente presentan pequeños cambios entre una imagen y la siguiente, con excepción de los cambios del fondo de la escena. Esta redundancia puede ser explotada transmitiendo únicamente las diferencias entre cuadros sucesivos, con lo que se logra una reducción en la tasa de datos. La ***codificación temporal o Interframe*** remueve las similitudes entre imágenes sucesivas mediante la codificación de diferencias. En las partes estáticas de la secuencia de imágenes, las diferencias temporales son cercanas a cero, por lo que no son codificadas. Por otro lado, las partes que cambian entre los cuadros, ya sea por cambios de iluminación o por el movimiento de objetos, generan un error de imagen significativo, el cual debe ser codificado.

Los cambios en la imagen debido al movimiento se pueden reducir significativamente si se estima el movimiento del objeto y la diferencia se toma con respecto a alguna imagen de referencia. La compresión temporal (Interframe) permite una reducción de aproximadamente 3:1 comparada con la compresión espacial (Intraframe) [6], [11].

2.6.1 Compensación y estimación de movimiento

La ***compensación de movimiento*** es una técnica utilizada en la compresión de video, cuyo objetivo es eliminar la redundancia temporal entre imágenes adyacentes en una secuencia de video para aumentar la compresión. La compensación de movimiento utiliza la ***estimación de movimiento***, un proceso que calcula la posición a la que se ha movido un objeto de un cuadro a otro [8].

Estimación de movimiento

La técnica de estimación de movimiento más utilizada en la compresión de video es el *algoritmo de igualación de bloques (BMA)*. En un algoritmo BMA típico, un cuadro de video se divide en bloques de MxN o N^2 pixeles. Cuando hay movimiento, con un desplazamiento máximo de w pixeles por cuadro, el bloque de pixeles del cuadro actual (*bloque de referencia*) se compara con otros bloques, empezando con el que tiene las mismas coordenadas, pero del cuadro previo, dentro de una ventana cuadrada de ancho $N + 2w$, como se ilustra en la figura 2.14 [11].

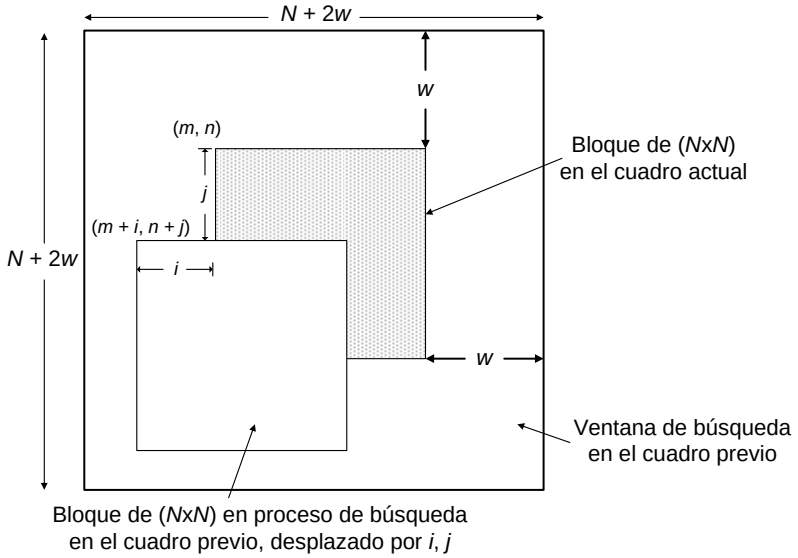


Figura 2.14 Cuadros previo y actual en una ventana de búsqueda

Una vez que se encuentra el bloque con mayor parecido al de referencia es posible cuantificar su movimiento. Las coordenadas del bloque con la mayor coincidencia se conocen como ***vectores de movimiento*** e indican el desplazamiento del bloque de referencia [11]**.**

Para obtener el bloque con mayor parecido al de referencia, se pueden utilizar métodos de comparación o medición como la *función de correlación cruzada (CCF)*, *el error cuadrático medio (MSE) y el error absoluto medio (MAE)*. Para lograr el objetivo, en la CCF la correlación tiene que ser maximizada, mientras que en las últimas dos medidas la distorsión debe de ser minimizada. En los codificadores prácticos se utilizan el MSE y el MAE, puesto que se cree que la CCF no da un buen seguimiento del movimiento, especialmente cuando el desplazamiento es muy pequeño. Las funciones de comparación del tipo MSE y MAE se definen en las ecuaciones (2.12) y (2.13) [11].

Para MSE:

$$M(i,j) = \frac{1}{N^2}\sum_{m=1}^{N}\sum_{n=1}^{N}(f(m,n) - g(m+i,n+j))^2, \qquad -w \le i,j \le w \tag{2.12}$$

y para MAE:

$$M(i,j) = \frac{1}{N^2}\sum_{m=1}^{N}\sum_{n=1}^{N}\left|f(m,n) - g(m+i,n+j)\right|, \qquad -w \le i,j \le w \tag{2.13}$$

donde: $f(m,n)$ es el bloque actual de N^2 pixeles en las coordenadas (m,n)

$g(m+i, n+j)$ es el bloque del cuadro previo en las nuevas coordenadas $(m+i, n+j)$

w indica el máximo desplazamiento de pixeles por cuadro

El bloque más parecido al bloque de referencia tendrá la posición $i = a$ y $j = b$. El vector de movimiento $MV(a,b)$ representará el desplazamiento de todos los pixeles dentro del bloque.

Compensación de movimiento

Una vez que se obtiene el vector de movimiento, al bloque de referencia se le resta el bloque con mayor parecido para producir un *bloque residual*, el cual es codificado y transmitido junto con el *vector de movimiento*. El decodificador utiliza el bloque residual y el vector de movimiento para reconstruir el bloque original [12].

2.6.2 Codificación de cuadros *I, P,* y *B*

Algunos estándares de compresión de video, como *MPEG (Grupo de Expertos de Imágenes en Movimiento)*, dividen cada secuencia de video en uno o más ***Grupos de imágenes (GOPs)*** y cada imagen del GOP la dividen en ***rebanadas*** de *macrobloques*. Un ***macrobloque*** consta de cuatro bloques de luminancia, un bloque de color U y otro de V. El ***bloque*** es la unidad básica para la compresión espacial, como se observa en la figura 2.15. El concepto de "rebanada" fue introducido por MPEG-1. Al dividir la imagen, si ocurre un error de datos dentro de una rebanada, éste no afecta a las demás rebanadas por lo que el resto del cuadro puede ser decodificado sin ningún problema [6].

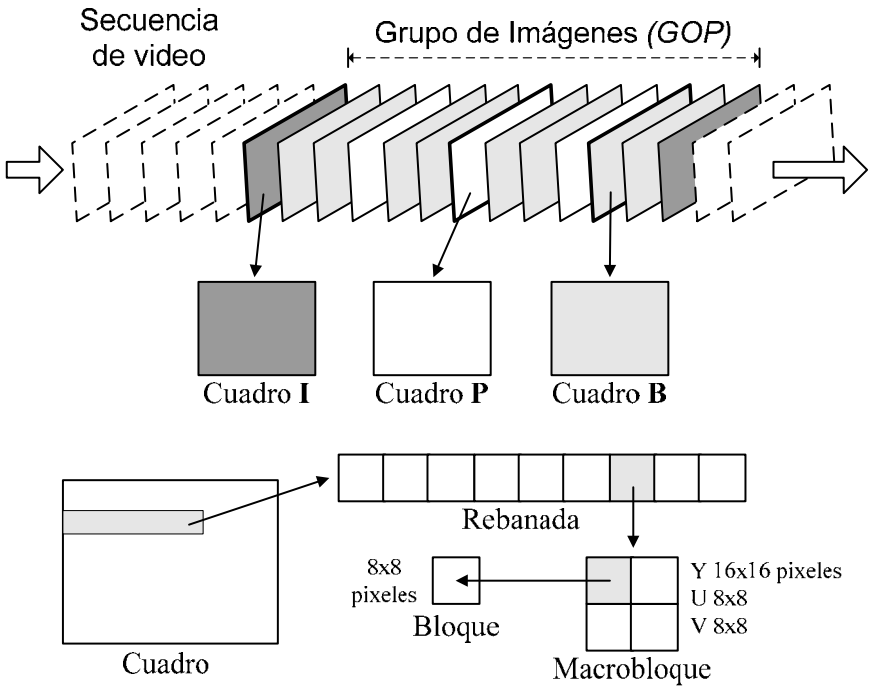


Figura 2.15 División de imágenes y tipos de cuadro en la compresión MPEG.

MPEG define tres tipos de cuadros dentro del grupo de imágenes: *cuadros I, P y B*. Cada GOP está compuesto de una o más imágenes o cuadros; una de estas imágenes debe ser un cuadro ***I***. Usualmente el espacio entre dos cuadros de referencia (Cuadros *I* o *P*) se representa como *M* y el espacio entre dos cuadros sucesivos ***I*** se representa como *N*. En la figura 2.15 se muestra como están ordenados los cuadros I, P, y B dentro de un grupo de imágenes [6].

Los ***cuadros I o Intraframe*** son codificados espacialmente y de manera independiente, es decir, únicamente se codifica la información dentro del cuadro sin tomar otros cuadros como referencia. Los cuadros *I* proporcionan puntos de referencia de acceso aleatorio en los datos de video comprimido y pueden ser decodificados independientemente sin referenciar otras imágenes. Con los cuadros *I* es factible tener un flujo de bits MPEG. Además, la propagación de errores debido a fallas de transmisión en imágenes previas termina en un cuadro *I*, dado que el cuadro *I* no utiliza ningún otro cuadro como referencia. Puesto que los cuadros *I* usan solamente codificación espacial, estos proporcionan una compresión moderada.

Los ***cuadros P o pronosticados*** se codifican utilizando compensación de movimiento de los cuadros *I* o *P* precedentes. Los cuadros *P* proporcionan una mayor compresión que los cuadros *I*, en virtud de que los primeros utilizan compensación de movimiento y están formados por vectores de movimiento. Los cuadros *P* sirven como referencia para cuadros *B* y *P* futuros; los errores de transmisión en los cuadros *I* y *P* se pueden propagar a las imágenes sucesivas, debido a que los cuadros *I* y *P* se utilizan para predecir las siguientes imágenes.

Los ***cuadros B o bidireccionales*** utilizan como referencia cuadros *I* o *P*, ya sean cuadros anteriores o posteriores con respecto al cuadro actual. Al hacer una interpolación entre los dos cuadros (previo y futuro) se obtiene la predicción de movimiento. Los cuadros B son los que proporcionan la mayor compresión [6], [15].

2.7 Codificación Entrópica

La ***codificación entrópica*** disminuye la redundancia entre los símbolos que representan una secuencia de video, usando técnicas de codificación de longitud variable, es decir utiliza palabras de código corto para las secuencias de bits más probables, y palabras de código largo para las secuencias de bits menos probables. Los símbolos a codificar pueden ser coeficientes transformados y cuantizados, vectores de movimiento, marcadores, encabezados o información suplementaria [11], [12].

Codificación de longitud variable

Para reducir más la tasa de bits, los coeficientes cuantizados y los vectores de movimiento se codifican mediante el método de ***codificación de longitud variable (VLC)***. En VLC, a los valores con la probabilidad más alta se les asigna palabras con código corto, mientras que a los valores menos probables se les asigna palabras con código largo; las longitudes de los códigos pueden variar inversamente con la probabilidad de ocurrencia de los diferentes símbolos. La tasa de bits requerida para codificar estos símbolos es el inverso del logaritmo de probabilidad, *p,* en base 2 (bits), por ejemplo $log_2\, p$. La ***entropía de los símbolos*** es el promedio mínimo de bits requerido para codificar los símbolos de los elementos de la secuencia de video y se puede calcular utilizando la ecuación (2.14) [11]:

$$H(x) = -\sum_{i=1}^{N} p_i \log_2 p_i \qquad (2.14)$$

Existen dos tipos de codificación de longitud variable empleadas en los codecs de video estándar: la ***codificación Huffman*** y la ***codificación aritmética***. Cabe hacer notar que la codificación Huffman es un simple código VLC, pero su compresión nunca será tan baja como la entropía debido a la restricción de que los símbolos asignados deben tener un número de bits entero. Sin embargo, con la codificación aritmética se puede aproximar a la entropía, puesto que los símbolos no son codificados individualmente. La *codificación Huffman* se emplea en muchos estándares de video para codificar los coeficientes de la transformada cuantizados, así como los vectores de movimiento. Por su parte, la codificación aritmética se emplea, por ejemplo, en estándares como JPEG2000 y en la codificación de imágenes y figuras de MPEG-4, en los que se requiere una compresión extra [11].

2.7.1 Codificación Huffman

La codificación Huffman está basada en probabilidades estadísticas. Esta codificación asigna un código de salida a cada símbolo, donde estos códigos pueden ser tan cortos como un bit, o considerablemente más largos que los símbolos de entrada, dependiendo de su probabilidad. El número óptimo de bits por cada símbolo esta dado por la ecuación (2.15) [11].

$$No.\ óptimo\ de\ bits\ por\ símbolo = -\log_2 p \qquad (2.15)$$

donde: p es la probabilidad de un símbolo dado.

Sin embargo, dado que las palabras de código asignadas requieren un número de bits entero, la codificación Huffman se vuelve sub-óptima. Por ejemplo, si la probabilidad de un símbolo es 0.33, el número óptimo de bits para codificar este símbolo será aproximadamente 1.6 bits, pero el esquema de codificación Huffman tiene que asignar uno o dos bits al código. En cualquier caso, el promedio del símbolo anterior utilizará más bits en comparación con su entropía. Como la probabilidad de un símbolo llega a ser muy alta, la codificación Huffman se hace poco óptima. Por ejemplo, para un símbolo con una probabilidad de 0.9, el tamaño de código optimo es 0.15 bits, pero la codificación Huffman asigna un valor mínimo de código de un bit al símbolo, el cual es 6 veces mayor que lo necesario. Por esta razón, los recursos son poco aprovechados [11].

Para generar los códigos Huffman de los símbolos con una probabilidad de ocurrencia conocida, se lleva a cabo el siguiente procedimiento [4]:

- Se clasifican todos los símbolos, de acuerdo a su probabilidad de ocurrencia, en orden decreciente.

- Se crea un árbol combinando los nodos en pares, empezando con los nodos de menor probabilidad. La combinación de dos nodos produce un nuevo nodo, cuya probabilidad es la suma de probabilidades de los nodos que lo han construido. Se continúa con este procedimiento hasta obtener un solo nodo.

- Para asignar los códigos a los símbolos, basta con recorrer el árbol desde la raíz a cada uno de los símbolos, asignando un 0 o un 1 dependiendo si se toma una rama hacia arriba o hacia abajo (la asignación de ceros y unos a las ramas puede ser totalmente arbitraria).

El procedimiento anterior resulta más claro si se ilustra con un ejemplo sencillo, como el que se muestra en la figura 2.16. Se puede observar como los nodos se van combinando de dos en dos, eligiendo siempre los de probabilidad más pequeña. Si existen varios nodos con la misma probabilidad, se pueden tomar dos nodos cualesquiera para combinarse. La probabilidad asignada al nodo raíz (identificado en la figura 2.16 con un punto negro) siempre debe ser uno.

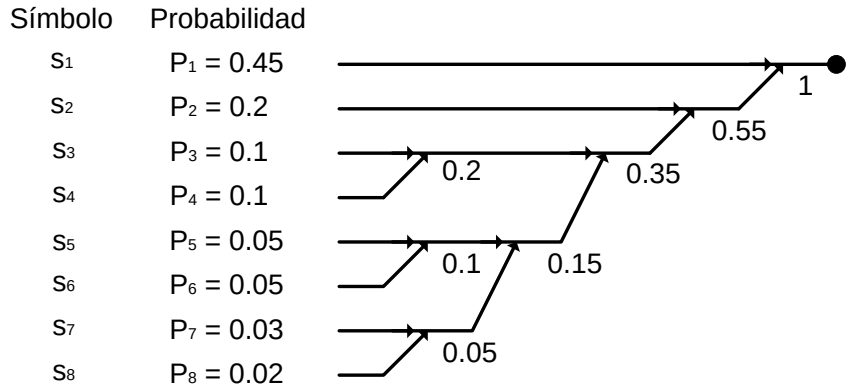


Figura 2.16 Proceso de creación del código Huffman.

En la figura 2.17 se muestra el árbol de la figura 2.16, pero ahora con la asignación de códigos a cada símbolo. Así, para determinar el código que se debe asignar al símbolo s_3, se debe partir del nodo raíz realizando dos pasos hacia abajo y después dos pasos más hacia la izquierda. El código resultante para este símbolo es 1100. Nótese que si seguimos esta construcción del árbol de Huffman, el símbolo con menor probabilidad siempre quedará codificado con valores igual a 1 [4].

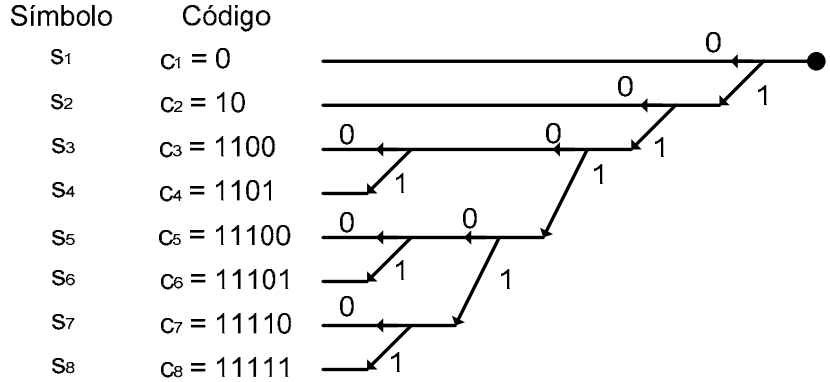


Figura 2.17 Proceso de generación de código a partir del árbol de Huffman.

En la tabla 2.3 se muestran las probabilidades de los símbolos del ejemplo anterior, sus códigos Huffman y el número de bits por cada código. El *promedio de bits por símbolo* se calcula mediante la suma de productos de la probabilidad de cada símbolo por el número de bits utilizados en su código.

Símbolo	Probabilidad	Código	Número de bits
s_1	0.45	0	1
s_2	0.2	10	2
s_3	0.1	1100	4
s_4	0.1	1101	4
s_5	0.05	11100	5
s_6	0.05	11101	5
s_7	0.03	11110	5
s_8	0.02	11111	5

Tabla 2.3 Probabilidad de los símbolos y número de bits utilizados por cada código.

El *promedio de bits por símbolo* es:
$0.45 \times 1 + 0.2 \times 2 + 0.1 \times 4 + 0.1 \times 4 + 0.05 \times 5 + 0.05 \times 5 + 0.03 \times 5 + 0.02 \times 5 = \textit{2.4 bits}$

el cual es muy cercano a la entropía de los datos originales, la cual se calcula usando la ecuación (2.14).

$H(x) = -(0.45log_20.45 + 0.2log_20.2 + 0.1log_20.1 + 0.1log_20.1 + 0.05log_20.05 + 0.05log_20.05 + 0.03log_20.03 + 0.02log_20.02) = \textit{2.344 bits}$

2.7.2 Decodificación Huffman

Una de las características de los códigos Huffman es que se puede decodificar la secuencia de datos a partir de la tabla de códigos sin necesidad de señalizar el principio y el fin de un símbolo. Para ilustrar el procedimiento de decodificación consideremos la cadena de bits 1110111010010111111...., cuyo proceso de decodificación se representa esquemáticamente en la tabla 2.4 [4].

Tabla de códigos	Secuencia de bits
$c_1 = 0$	1110111010010111111...
$c_2 = 10$	1 1 1 0 1 ⟶ $c_6 = 11101$
$c_3 = 1100$	
$c_4 = 1101$	**11101**11010010111111...
$c_5 = 11100$	1 1 0 1 ⟶ $c_4 = 1101$
$c_6 = 11101$	
$c_7 = 11110$	**11101110**1001011111...
$c_8 = 11111$	0 ⟶ $c_1 = 0$

Tabla 2.4 Ejemplo de decodificación de una secuencia de símbolos.

Al inicio de la decodificación, se comprueba si el primer bit (1) es una palabra código, buscándolo en la tabla de códigos con un solo bit (tabla 2.4). Si no coincide con ningún código, se comprueba si los dos primeros bits de la secuencia (11) pueden ser una palabra código, y se continua con este procedimiento hasta que se encuentra la palabra 11101, la cual se encuentra en la tabla 2.4 y corresponde al símbolo s_6. Siguiendo este procedimiento se puede decodificar la secuencia de símbolos completa que corresponde a los códigos c_6, c_4, c_1, c_1, c_2 y c_8 [4].

Por otra parte, cuando se produce un error en un bit, se puede alterar alguna palabra código, lo que supone la pérdida del sincronismo y la posibilidad de que aparezcan varios errores consecutivos. Sin embargo, el receptor puede detectar de forma automática la aparición de errores, si no encuentra las palabras código en las tablas o si la estadística de aparición de símbolos no corresponde con la esperada. Los algoritmos para la detección de errores y la sincronización con la cadena de datos son sumamente complejos.

En aplicaciones de almacenamiento de datos es recomendable que la codificación y la decodificación se realicen empleando *buffers temporales de memoria*, con una longitud que sea múltiplo de 8 bits. Con esto, cuando el buffer está lleno, se pueden almacenar la secuencia de bits, aprovechando una palabra de un byte o múltiplo de un byte [4].

Se puede apreciar que para un gran número de símbolos, como los valores de los coeficientes de la DCT, la codificación Huffman puede generar una larga cadena de bits para los valores que ocurren muy raramente, lo que resulta impráctico. En tales casos, un grupo de símbolos se representa por su conjunto de probabilidades y éstas se codifican por Huffman. A este método se le llama código Huffman modificado y es utilizado en el estándar de compresión de imágenes JPEG. Otro método, el cual es usado por MPEG entre otros, es la codificación Huffman bidimensional [11].

2.8 Estándares de compresión de video

Los codecs de compresión de video se pueden clasificar en:

- Estándares internacionales
- Formatos de propietarios
- Estándares abiertos

Los estándares internacionales frecuentemente utilizan tecnología patentada. Una autoridad de licencias controla la recolección de las cuotas en nombre de los propietarios de la patente. Los códigos abiertos son libres para todo usuario [6].

Los codecs de video más utilizados actualmente tienen su origen en las industrias de telecomunicaciones y multimedia. Estos son algunos de los codecs más populares:

H.261

Este fue el codec de video original, y el punto de partida para el estándar MPEG-1. H.261 fue desarrollado bajo el auspicio de la *Unión Internacional de Telecomunicaciones (ITU)* para videófonos y videoconferencias sobre *Redes Digitales de Servicios Integrados (ISDN)*. El videófono está empezando a ser un producto viable, mientras que la videoconferencia se ha convertido en un medio de comunicación clave en el mundo de los negocios. Una de las demandas requeridas para ambas aplicaciones es la operación en tiempo real, de tal manera que los codecs deben de tener un tiempo de procesamiento muy corto.

El estándar H.261 usa los formatos CIF, 352x288 pixeles, y cuarto de CIF (QCIF) de 176 x 144 pixeles, además, utiliza escaneo progresivo y submuestreo 4:2:0. H.261 soporta tasas de datos desde 64 kbit/s hasta 2 Mbits/s. La compresión se basa en la DCT con codificación de longitud variable. Opcionalmente, se pueden utilizar cuadros de predicción intermedia (P) con predicción de movimiento [6].

MPEG-1

MPEG-1 fue el primer estándar exitoso desarrollado por la comunidad multimedia para codificación audio visual. Este estándar ha sido utilizado por mucho tiempo para almacenamiento y reproducción de video en *Disco Compacto (CD)*. La resolución normal que utiliza es el formato SIF. La resolución espacial es diferente para sistemas PAL y NTSC (352x288 a 25 fps para PAL, y 352x240 a 30 fps para NTSC). MPEG-1 utiliza escaneo progresivo. La compresión de video, al igual que H.261, utiliza la DCT con codificación de longitud variable. La predicción de movimiento fue mejorada sobre H.261 con vectores de movimiento de subpixeles y la introducción de cuadros bidireccionales estimados (B). MPEG-1 fue diseñado para aplicaciones de almacenamiento como CD, con tasas de datos arriba de 1.5 Mbits/s y no es soportado para aplicaciones de transmisión [6].

H.263

H.263 es una evolución de H.261 dirigido a aplicaciones que requieren una tasa de bits baja. H.261 no tenía una tasa de datos lo suficientemente baja para operar a 28 kbit/s, así que no pudo ser utilizado para aplicaciones de videofonía sobre circuitos de telefonía analógica. El estándar H.263 soporta resoluciones SQCIF, QCIF, CIF, 4CIF y 16CIF y es el estándar de referencia para la codificación de video MPEG-4 [6].

MPEG-2

MPEG-2 fue planeado para remplazar los sistemas de televisión analógicos (NTSC, PAL) por sistemas de transmisión digital, con mayor resolución y una calidad más alta. Este estándar también es utilizado para codificación de DVD. Las primeras aplicaciones usaban anchos de banda mayores a 4 Mbits/s. El perfil principal es una tasa de cuadros y resolución para *Televisión de Definición Estándar* con tasas de datos arriba de los 15 Mbit/s. MPEG-2 fue extendido para soportar tasas de bits para Televisión de Alta Definición (arriba de 80 Mbit/s). MPEG-3 estaba planeado para ser un estándar de alta definición separado pero fue absorbido en favor de las extensiones de MPEG-2 y MPEG-4 [6].

MPEG-4

MPEG-4 es otro estándar desarrollado por la *Organización Internacional de Normalización (ISO)* y la *Comisión Electrotécnica Internacional (IEC)*, el cual pretende proporcionar herramientas y algoritmos para un almacenamiento eficiente, transmisión y manipulación de datos de video en ambientes multimedia.

El principal cambio en MPEG-4 es la *codificación basada en objetos*, donde una escena de video puede ser manejada de forma más óptima como un conjunto de objetos de primer plano y de fondo, que como una serie de cuadros rectangulares. Este tipo de codificación abre muchas posibilidades, como la codificación independiente de diferentes objetos en una escena, reutilización de los componentes de una escena, la composición (donde los objetos de varias fuentes son combinados dentro de una escena) y un alto grado de interactividad. MPEG-4 también utiliza la transformada Wavelet.

El concepto básico utilizado en MPEG-4 es el de *objeto de video (VO)*. Una *secuencia de video (VS)* está constituida por un número determinado de VOs. Por ejemplo, la escena mostrada en la figura 2.18 consiste de un VO de fondo y dos VOs de primer plano. MPEG-4 proporciona herramientas que habilitan cada objeto de video para ser codificado independientemente. El equivalente de un *cuadro* en términos de objetos de video, por ejemplo una foto de un VO en un instante de tiempo, es un *objeto de video plano (VOP)*. La escena entera puede ser codificada como un VOP rectangular único y este es equivalente a una imagen en términos de MPEG-1 y MPEG-2 [1].

Figura 2.18 Escena de video mostrando múltiples objetos de video.

AVC (Codec de Video Avanzado, H.264)

En muchas aplicaciones en los últimos diez años, H.263 no cumplió con el desempeño esperado en un codec de video. Los grupos MPEG e ITU realizaron de manera separada el desarrollo de nuevos codecs, así que se organizó un *equipo de video conjunto (JVT)* para desarrollar el *codec de video avanzado o AVC*. La realización de este trabajo fue un nuevo codec de alto desempeño para rivalizar con los mejores codecs de propietarios. H.264 fue diseñado por la ITU mientras que MPEG-4 por la organización MPEG [6].

MJPEG

El MJPEG *(JPEG en Movimiento)* es un método de compresión no estandarizado que codifica una secuencia de video como una serie de imágenes JPEG, donde cada una corresponde a un cuadro de video. El estándar JPEG es muy utilizado en la compresión de imágenes a color y en escala de grises. JPEG comprime el tamaño del archivo eliminando datos de la imagen de manera selectiva, es decir, utiliza compresión con pérdidas. El estándar también soporta una compresión sin pérdidas con una relación de reducción de datos de 3:1 aproximadamente, pero es más común que se utilice en el modo con perdidas con razones de 20:1 o mayores. La compresión JPEG está basada en la *Transformada Coseno Discreta (DCT)* aplicada en bloques 8x8, seguida por la cuantización, reordenamiento y codificación de longitud variable [6], [12].

En JPEG, una tasa de compresión más baja daría como resultado menos datos para representar a la imagen, pero el algoritmo de compresión degradaría algunos detalles finos de la imagen. Conforme se incrementa la compresión aparecen más artefactos y efectos de bloques indeseables. El nivel de calidad de las imágenes que es considerado aceptable depende de la aplicación [6].

Originalmente JPEG no estaba destinado para ser utilizado para compresión de video, sin embargo, MJPEG se ha vuelto muy popular y es utilizado en varias aplicaciones de transmisión y almacenamiento de video. Este codec no explota la redundancia temporal inherente en el movimiento en una secuencia de video por lo que su desempeño de compresión es pobre comparado con los codecs Interframe. Sin embargo, MJPEG presenta las siguientes ventajas:

- **Baja complejidad:** La complejidad del algoritmo, y los requisitos de hardware, procesamiento y almacenamiento son muy bajos comparados con algún codec Interframe básico (por ejemplo H.261).

- **Tolerancia de error:** La codificación Intraframe limita el efecto de error al procesar las imágenes de manera independiente, ya que si se presenta un error en una imagen, este no se propaga en las siguientes imágenes como en el caso de MPEG-1 y MPEG-2, los cuales son superados por MJPEG en ambientes ruidosos.

- **Posición en el mercado:** JPEG es quizás el estándar de compresión de imágenes más conocido y utilizado en el mercado, así que los usuarios potenciales ya están familiarizados con la tecnología de *JPEG en Movimiento*. Debido a su bajo desempeño en compresión, MJPEG solo es adecuado para comunicaciones con gran ancho de banda (como redes dedicadas, por ejemplo). Paradójicamente, los usuarios generalmente tienen una buena experiencia de MJPEG debido a que las instalaciones no tienden a sufrir de los problemas de retardos y ancho de banda encontrados en los codecs Interframe utilizados en redes como Internet o canales con tasas de bits bajas. Por otro lado, MJPEG es popular para aplicaciones como captura de video, edición de video en PC y cámaras de seguridad [1], [12].

En este trabajo se decidió utilizar el método de compresión MJPEG debido a que la compresión es muy factible de implementar por hardware, y además al procesar las imágenes de manera independiente, si se presenta un error en una imagen, este no se propaga en las siguientes imágenes. Otra de las ventajas de MJPEG que lo hace muy atractivo para sistemas de transmisión de video, es el hecho de que conforme se va adquiriendo una imagen correspondiente a una secuencia de video, se puede comprimir para transmitirse inmediatamente, lo que reduce los tiempos de ejecución para todo el sistema. En el *capítulo 3* se analiza detalladamente el método de compresión MJPEG.

MJPEG-2000

Otro método de compresión de video no estandarizado, similar a MJPEG, es el *MJPEG-2000 (JPEG-2000 en movimiento)*, donde la secuencia de video se comprime como una secuencia de cuadros individuales, aplicando a cada cuadro la compresión JPEG-2000 [6].

JPEG-2000 fue desarrollado para proporcionar mayor eficiencia que el original JPEG. Este utiliza la *Transformada Wavelet Discreta (DWT)* como base para su método de codificación. JPEG2000 proporciona un desempeño de compresión superior a JPEG y no exhibe las características de efectos indeseables que se presentan en los métodos de compresión basados en la DCT.

El estándar JPEG original ha ganado mucha aceptación y ahora se encuentra en muchas aplicaciones de computación. Sin embargo, los algoritmos basados en la DCT tienen algunas desventajas, donde quizás la más importante es la presencia de artefactos indeseables en las imágenes con una compresión JPEG muy alta. Desde su aparición, se han desarrollado muchos esquemas de codificación alternativos para superar al estándar JPEG. La necesidad de un mejor desempeño en tasas de compresión alta llevó al desarrollo del estándar JPEG-2000. Las características de JPEG-2000 son las siguientes:

- Desempeña una buena compresión, particularmente en tasas de compresión altas.

- Compresión eficiente de imágenes de tono continuo, de dos niveles y compuestas (por ejemplo una imagen fotográfica con texto sobrepuesto).

- Compresión con pérdidas y sin pérdidas (dentro de la misma estructura).

- Transmisión progresiva. JPEG-2000 soporta escalabilidad *SNR (Señal a ruido)*.

- Codificación de *ROI (Regiones de interés).* Esta característica permite al codificador especificar una región arbitraria dentro de la imagen que puede ser tratada de manera distinta durante la codificación, por ejemplo codificando una región especificada con una calidad más alta o permitiendo una decodificación independiente de las ROIs.
- Las herramientas de recuperación de errores incluyen particionamiento de datos, detección de error y ocultamiento.

- Arquitectura abierta. El estándar JPEG-2000 proporciona una estructura abierta la cual puede hacer relativamente fácil añadir nuevas características de codificación, ya sea como parte del estándar o como un complemento del estándar.

La arquitectura de un codificador JPEG-2000 se muestra en la figura 2.19. Esta es aparentemente similar a la arquitectura JPEG, pero una diferencia importante es que la misma arquitectura puede ser utilizada para codificación con perdidas o sin perdidas [1].

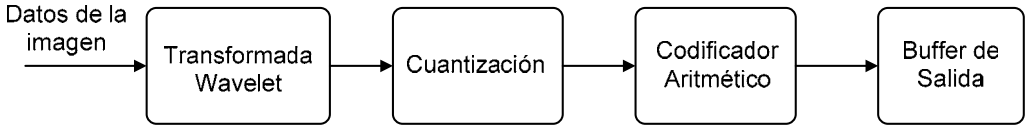


Figura 2.19 Arquitectura de un codificador JPEG-2000.

La unidad de codificación básica de JPEG-2000 se denomina *"tile"*; ésta es normalmente una región de la imagen de 2^n x 2^n. La imagen está compuesta por *tiles* del mismo tamaño que no se superponen. Cada tile es codificado como sigue:

Transformada: A cada *tile* se le aplica la *transformada Wavelet* para descomponerlos en una serie de sub-bandas. La transformación puede ser reversible (para aplicaciones de codificación sin perdidas) o irreversible (adecuada para aplicaciones de codificación con perdidas).

Cuantización: Los coeficientes de la transformada Wavelet son cuantizados de acuerdo a la importancia de cada sub-banda para la apariencia final de la imagen. Existe una opción para dejar los coeficientes sin cuantizar (codificación sin pérdidas).

Codificación Entrópica: JPEG-2000 utiliza una forma de codificación aritmética para codificar los coeficientes cuantizados antes de la transmisión o almacenamiento. La codificación aritmética puede proporcionar una compresión más eficiente que la codificación de longitud variable.

El resultado es un estándar de compresión con un desempeño de compresión significativamente mejor que JPEG. Para la misma calidad de imagen, JPEG-2000 puede comprimir imágenes al menos dos veces más que JPEG. En tasas de compresión altas, la calidad de las imágenes se degrada, y la imagen decodificada muestra un efecto gradual de borrosidad que mejora el efecto de bloques más obvio asociado con la DCT. Esta ganancia en el desempeño de la compresión se logra a costa de incrementar la complejidad y los requisitos de almacenamiento durante la codificación y decodificación, pero las imágenes toman más tiempo para almacenarse y desplegarse utilizando JPEG-2000 [1].

RESUMEN

En este capítulo, se han expuesto los fundamentos de compresión de video digital, tanto de compresión con pérdidas, como sin pérdidas, siendo la primera de nuestro interés, ya que es la que proporciona una mayor compresión. También se expuso la necesidad y los beneficios de la compresión en los sistemas de video digital. Además, se hizo una descripción de los tipos de redundancia presentes en el video digital que pueden ser explotadas para lograr la compresión.

Por otro lado, se estudió la Codificación Espacial o Intraframe, al igual que algunos métodos que la componen, como la codificación DPCM y la codificación por transformadas. También se hizo el análisis de un proceso muy importante en la compresión de video, que es la cuantización. En este capítulo, se hizo un análisis de la Codificación Interframe, con una descripción de la compensación de movimiento y de la codificación de cuadros I, P y B en un grupo de imágenes. De manera similar se estudiaron los principios de codificación entrópica, la cual es necesaria para reducir la redundancia entre símbolos.

Finalmente, se describieron las características de los estándares y métodos de compresión más populares en la compresión de video digital, así como sus ventajas y desventajas. El algoritmo MJPEG es el que presenta menor complejidad, y los requisitos de hardware, procesamiento y almacenamiento son muy bajos comparados con algún otro codec Interframe. Además, MJPEG limita el efecto de error al procesar las imágenes de manera independiente, ya que si se presenta un error en una imagen, este no se propaga en las siguientes imágenes como en el caso de MPEG-1 y MPEG-2. MJPEG supera a los codecs Interframe en ambientes ruidosos. Por esta razón, se decidió utilizar MJPEG en la implementación de nuestro sistema de adquisición y compresión de video utilizando el DSP C6416 de Texas Instruments.

3.- COMPRESION DE VIDEO MJPEG

Dado que la transmisión y almacenamiento de video requieren una gran cantidad de información, es necesario el uso de la compresión de video, con la cual se reduce de forma importante la cantidad de datos a transmitir o almacenar.

Este capítulo tiene como objetivo describir el método de compresión de video MJPEG, el cual se implementa en este trabajo. Como el método MJPEG está basado en el estándar de compresión de imágenes JPEG, en este capítulo se describen los modos de codificación JPEG y las etapas necesarias para efectuar la compresión y descompresión de imágenes, así como algunos de los marcadores más importantes en el estándar.

3.1 Método de compresión MJPEG

MJPEG es un método de compresión no estandarizado que utiliza la compresión JPEG para cada cuadro de video. Esto proporciona acceso aleatorio a los cuadros de video de manera individual, sin embargo las tasas de compresión son demasiado bajas, debido a que la técnica no toma ventaja de la redundancia temporal entre cuadros adyacentes [17].

Aplicaciones

MJPEG es usado frecuentemente en sistemas de edición de vídeo, además, MJPEG es utilizado en videocámaras IP y en video conferencias de baja calidad en tiempo real. Para reproducir este formato con una tasa de cuadros por segundo considerable se requiere de la codificación-decodificación JPEG [4].

Ventajas

- MJPEG soporta acceso de cuadros aleatorio ya que no se utiliza codificación *Interframe*.
- Al utilizar solamente compresión *Intraframe,* la compresión es independiente de la cantidad de movimiento en la escena, ya que no se utiliza predicción temporal.
- Al procesar las imágenes de manera independiente se limita el efecto de error, ya que si se presenta un error en una imagen, éste no se propaga en las siguientes imágenes como en el caso de MPEG-1 y MPEG-2.
- MJPEG proporciona tasas de compresión entre 10:1 y 15:1 sin afectar significativamente la calidad visual de las imágenes.
- MJPEG da buenos resultados en tiempo real con equipos de bajo costo [17].
- La implementación de MJPEG es relativamente simple, la complejidad del algoritmo, los requisitos de hardware, procesamiento y almacenamiento son muy bajos en comparación con los codecs *Interframe* [1], [12].

Desventajas

- MJPEG no explota la redundancia temporal de movimiento en una secuencia de video por lo que su desempeño de compresión es bajo comparado con los codecs *Interframe*.

- MJPEG requiere de un gran ancho de banda para su transmisión.

- Los archivos MJPEG son de gran tamaño, por lo que requieren mayor espacio de almacenamiento.

- MJPEG no está estandarizado, por lo que puede haber problemas de compatibilidad con los archivos de salida de distintos fabricantes.

- Conforme se incrementa la compresión, aparecen más efectos de bloque indeseables [6].

Dado que el método de compresión MJPEG está basado en el estándar de compresión de imágenes JPEG en las siguientes secciones nos enfocaremos en el estudio de este estándar.

3.2 Descripción del estándar JPEG

El *estándar internacional ISO 10918-1* conocido por el acrónimo del grupo que lo desarrolló, el *Joint Photographic Experts Group (Unión Grupal de Expertos en Imágenes)* surgió en 1992 y es el estándar que proporciona un método y sintaxis para compresión de imágenes de tono continuo (como imágenes fotográficas). El codificador JPEG está orientado tanto a imágenes monocromáticas como a color y puede considerarse como un conjunto de herramientas que admite diferentes modos y estrategias de compresión en función de las características del sistema en el que se debe utilizar.

La principal aplicación del estándar JPEG está en la transmisión y almacenamiento de imágenes comprimidas y es ampliamente utilizado en compresión de imágenes digitales, cámaras digitales, imágenes en páginas web y muchas aplicaciones más. Aunque JPEG fue diseñado con la intención de comprimir imágenes fijas, encontró gran popularidad como un método simple y efectivo de compresión de video en la forma de *MJPEG* [1].

Los objetivos que se propuso la comisión encargada de definir el estándar JPEG pueden sintetizarse en los siguientes puntos:

a) Conseguir codificadores y decodificadores cuyas tasas de compresión y calidad de imagen estuvieran a la altura de la tecnología actual.

b) Los métodos propuestos debían ser útiles para codificar imágenes monocromáticas en niveles de gris, imágenes de color o en bandas distintas al espectro óptico (por ejemplo imágenes obtenidas por satélites de inspección). Además, los métodos presentados debían ser independientes del tamaño y resolución espacial de la imagen.

c) La implementación de los métodos debían soportar su realización mediante *software*, que pudiera ejecutarse en múltiples plataformas de manera rápida y eficiente. Los algoritmos también debían soportar su implementación mediante circuitos integrados dedicados, de bajo o medio costo.

Los resultados y la difusión del estándar JPEG confirman que los objetivos planteados para este estándar se han logrado exitosamente [4].

Archivos JPEG

Uno de los principales problemas del estándar JPEG es que éste no definió desde un principio un *formato de archivo*. El estándar JPEG no especifica lo que necesita una aplicación para crear imágenes que puedan ser intercambiadas entre diferentes aplicaciones. Por ejemplo, JPEG no dice nada sobre como deben ser representados los colores, solo menciona como se deben almacenar los valores de los componentes de color. Tampoco existe alguna definición de como se deben de mapear los valores de los componentes dentro de un espacio de color, entre otros inconvenientes [3].

Naturalmente, se tuvo que crear una solución, y esta llegó con el *Formato de Intercambio de Archivos JPEG (JFIF)*, creado por Eric Hamilton. Esta especificación llena los huecos dejados por JPEG en la creación de archivos que pueden ser intercambiados entre aplicaciones. *JFIF* se ha convertido en el sinónimo de *Archivo JPEG*. Mientras otros formatos de archivos, como el *Formato de archivo de imágenes etiquetada (TIFF)*, utilizan compresión JPEG, un archivo con extensión JPG o JPEG invariablemente tiene el formato JFIF [3].

Orden de los bytes en un archivo JPEG

En las computadoras cada byte se identifica con su posición en la memoria. Cuando se manejan números de más de un byte, éstos también deben estar ordenados. Este aspecto es particularmente importante en la programación en código máquina, ya que algunas máquinas consideran el byte situado en la dirección más baja el byte menos significativo (arquitectura *little endian*, como los procesadores Intel) mientras que otras consideran que ése es el más significativo (arquitectura *big endian*, como los procesadores Motorola). De este modo, un byte con el número decimal 27 se almacenaría en una máquina *little endian* igual que en una máquina *big endian*, ya que sólo ocupa un byte. Sin embargo, para números más grandes, los bytes que los representan se almacenarían en distinto orden en cada arquitectura.

Por ejemplo, consideremos el número hexadecimal entero AABBCCDD, de 32 bits (4 bytes), localizado a partir de la dirección 100 de la memoria. El número ocuparía las posiciones desde la localidad 100 a la 103, pero dependiendo de si la máquina es little o big endian, los bytes se almacenarían de diferente manera, como se muestra en la figura 3.1:

Little-endian (Como Intel)

Dirección	...	100	101	102	103	...
Dato	...	DD	CC	BB	AA	...

Big-endian (Como Motorola)

Dirección	...	100	101	102	103	...
Dato	...	AA	BB	CC	DD	...

Figura 3.1 Almacenamiento en memoria para una máquina little-endian y big-endian

Algunas arquitecturas de microprocesador tales como ARM, Power PC, DEC Alpha, PARISC y MIPS pueden trabajar con ambos formatos, y a veces son referidas como sistemas *middle-endian.*

El formato en el que se almacenan los datos en un archivo JPEG es el formato *big-endian*. Las cadenas de bits están codificadas dentro de bytes empezando por los bits más significativos. Cuando una cadena de bits cruza un límite de byte, los bits en el primer byte son más significativos que los bits del segundo byte. El DSPC6416, el cual es utilizado en este trabajo para realizar la compresión, se puede utilizar tanto el formato little-endian como big-endian [3].

Frecuencia de muestreo en JPEG

En JPEG, la *frecuencia de muestreo* es la frecuencia relativa en la que un componente es muestreado. En muchos formatos de archivos gráficos, todos los componentes de color están muestreados a la misma frecuencia. JPEG permite que los componentes individuales puedan ser muestreados a diferentes frecuencias (submuestreo). Las frecuencias de muestreo permiten que las imágenes sean comprimidas variando la cantidad de información con la que contribuye cada componente. Como se vio en el *capítulo 1*, en el espacio de color YCrCb el componente Y es el más importante. La reducción de la cantidad de información de los componentes Cr y Cb es un método simple para reducir el tamaño de la imagen [3].

3.3 Modos de compresión JPEG

El estándar JPEG define cuatro modos de compresión: secuencial, progresivo, sin pérdidas y jerárquico. Adicionalmente, el estándar define múltiples procesos de codificación para cada modo. La tabla 3.1 muestra la relación entre los modos de compresión JPEG y los procesos de codificación. Mientras existen algunas características en común entre ellos, la mayoría deben ser implementados con técnicas completamente diferentes [3].

JPEG										
Secuencial				*Progresivo*				*Sin pérdidas*		*Jerárquico*
Huffman		Aritmética		Huffman		Aritmética		Sin pérdidas original	JPEG-LS	
8 bits	12 bits	8 bits	12 bits	8 bits	12 bits	8 bits	12 bits			

Tabla 3.1 Modos de compresión JPEG.

Modo de codificación secuencial

Este modo define los métodos utilizados para comprimir las imágenes siguiendo un orden de codificación de los bloques de izquierda a derecha y de arriba abajo. El tamaño de los bloques es de 8x8 muestras. El decodificador recupera la imagen en el mismo orden. El modo secuencial soporta muestras de datos con 8 y 12 bits de precisión. Por otro lado, dentro del modo secuencial, existen dos alternativas de codificación entrópica: una utilizando codificación Huffman y la otra usando codificación aritmética. Los archivos JPEG más populares utilizan el modo secuencial con codificación Huffman y muestras de datos de 8 bits. En la figura 3.2 se muestra la reconstrucción de una imagen en modo secuencial [3], [11].

Figura 3.2 Reconstrucción de una imagen utilizando el modo secuencial.

El estándar JPEG también define otro proceso de codificación secuencial que utiliza codificación Huffman. Este es el proceso denominado *modo básico* o *baseline*, el cual es un subconjunto del modo secuencial con codificación Huffman y está diseñado para adaptarse a un amplio número de aplicaciones. Un decodificador que pueda manejar el modo extendido es capaz de manejar el modo básico de manera transparente. *Las imágenes codificadas en modo básico solo pueden tener muestras de 8 bits, y sus tablas de cuantización y Huffman son más pequeñas que en las imágenes en modo secuencial extendido*.

Para el desarrollo de este trabajo se eligió el modo básico secuencial del estándar JPEG, debido a que es el algoritmo utilizado en todos los modos de codificación JPEG y es la base para la codificación de las imágenes *Intraframe* en el estándar MPEG [3],[4].

Modo de codificación progresivo

En el modo progresivo, la imagen se codifica en varias exploraciones utilizando siempre la misma resolución espacial. El decodificador puede obtener, de forma rápida, una primera aproximación de la imagen, la cual se clarifica al avanzar el proceso de decodificación. Esto es particularmente útil para el acceso a imágenes localizadas en centros remotos en los que las comunicaciones son lentas o en buscadores web, debido a que permiten que el usuario tenga una idea de lo que contiene la imagen después de que se han transmitido unos pocos datos [3], [4].

JPEG soporta dos versiones de codificación progresiva: por ***selección espectral***, donde cada escaneo toma un subconjunto de los coeficientes de la DCT de cada bloque (por ejemplo, solo coeficientes DC, coeficientes AC de baja frecuencia y coeficientes AC de alta frecuencia) y por ***aproximaciones sucesivas***, donde el primer escaneo contiene los *N* bits más significativos de cada coeficiente y el último escaneo los bits menos significativos. Un *escaneo* es una exploración única de los datos para uno o más componentes de la imagen.

La figura 3.3 muestra una imagen codificada y decodificada por selección espectral. La primera imagen contiene los coeficientes DC de cada bloque, la segunda imagen contiene los coeficientes DC y dos de los coeficientes más bajos de AC y la tercera contiene todos los 64 coeficientes en cada bloque. Se puede observar que con cada *escaneo* la imagen se empieza a aclarar [1], [19].

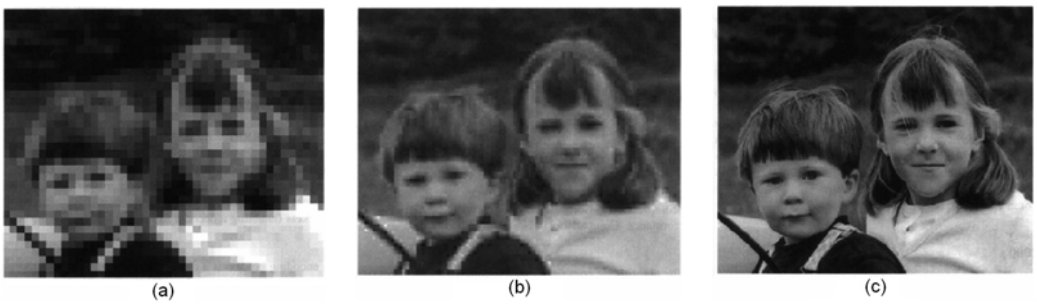


Figura 3.3 Ejemplo de codificación progresiva por selección espectral: (a) Solo con coeficientes DC; (b) coeficientes DC + dos de AC; (c) Todos los coeficientes.

El modo progresivo presenta la desventaja de que es mucho más difícil de implementar que el modo secuencial y que, si la imagen se está viendo conforme se está descargando, esto requiere mucho más procesamiento. El modo progresivo JPEG es más adecuado cuando la potencia de procesamiento excede a la velocidad de transmisión de la imagen [3].

Modo de codificación jerárquica

Este es un modo super-progresivo en el cual la imagen se divide en una serie de sub-imágenes llamados también *cuadros o frames*. Cada cuadro es una colección de uno o más escaneos. En el modo jerárquico, la imagen se codifica con distintos niveles de resolución, como se muestra en la figura 3.4. La resolución más baja es la primera que se obtiene en el decodificador. A partir de esta imagen preliminar puede irse decodificando el resto de datos hasta obtener la resolución deseada. El proceso de decodificación puede interrumpirse en cualquier momento [3], [11].

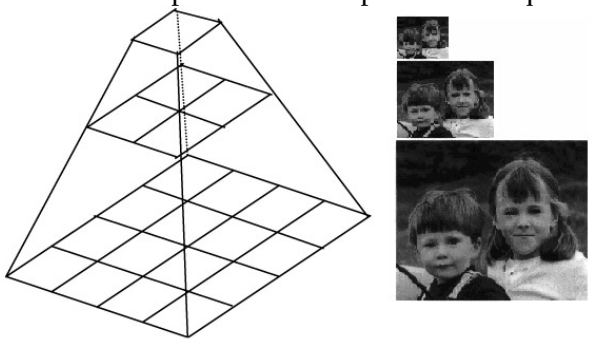


Figura 3.4 Codificación de una imagen con varias resoluciones en modo jerárquico.

Los defensores del modo jerárquico sostienen que este es mejor que el modo progresivo cuando las tasas de transmisión utilizadas son bajas. Si sólo se desea una imagen de baja resolución, no se requieren de todos los cuadros para obtener el resultado deseado. El principal inconveniente del modo jerárquico es que su implementación tiene mayor complejidad que los otros modos. El modo jerárquico JPEG requiere mucho más procesamiento que los otros modos y el uso de múltiples cuadros aumenta la cantidad de los datos que deben ser transmitidos [3].

Modo de codificación sin pérdidas

Originalmente el estándar JPEG definió un modo de codificación sin pérdidas basado en la codificación diferencial DPCM y en estructuras de predicción simples (en este modo no se utiliza la DCT). Este modo de codificación obtiene factores de compresión relativamente bajos, además para la mayoría de aplicaciones no comprime tan bien como otros formatos disponibles en el mercado, por lo que no hay una razón suficientemente fuerte para utilizar este formato de codificación. En los últimos años, se ha creado un nuevo método de compresión sin pérdidas conocido como JPEG-LS que, para todos los propósitos prácticos, ha hecho que el formato original JPEG sin pérdidas sea obsoleto [3].

3.4 Proceso de codificación JPEG en modo básico

La figura 3.5 muestra el esquema general de compresión JPEG en el modo básico. Los datos de la imagen son procesados en bloques de 8x8 muestras llamados *unidades de datos*. Los componentes de color (por ejemplo R, G, B ó Y, Cr, Cb) pueden ser procesados de forma separada (un componente completo a la vez) o en forma entrelazada (por ejemplo, un bloque compuesto por muestras de cada uno de los tres componentes de color en orden sucesivo). A continuación se describe cada etapa del codificador [1].

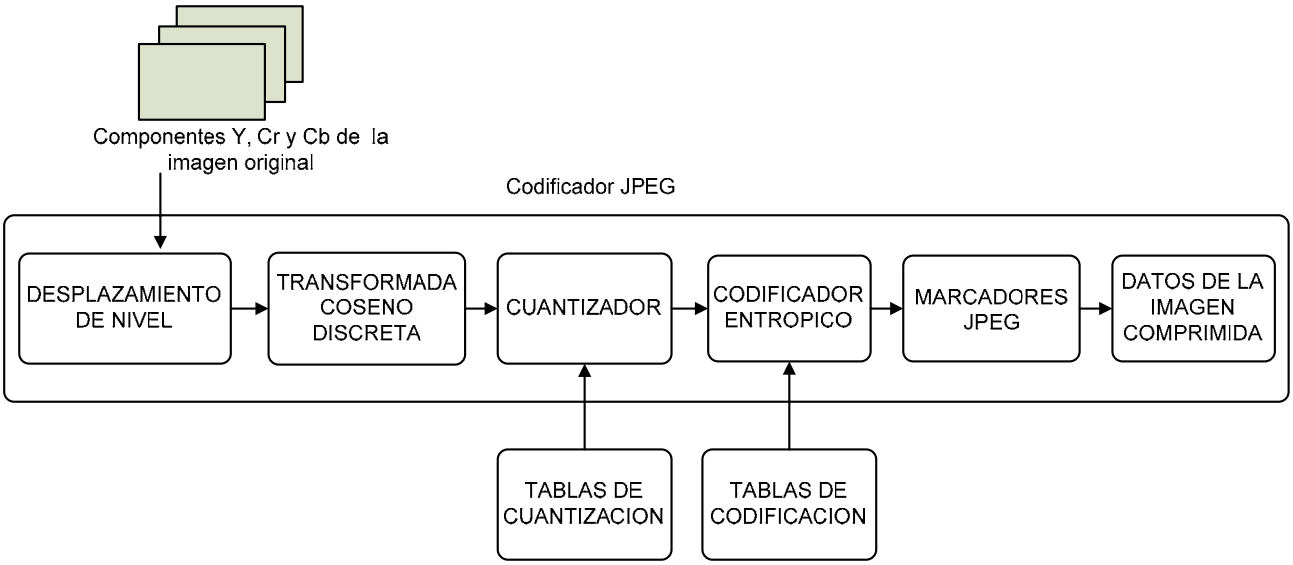


Figura 3.5 Esquema general de compresión JPEG.

Unidad de datos y Unidad Mínima de Codificación (MCU)

Una *unidad de datos* es el conjunto de datos más pequeño de la imagen que puede ser procesada en algún modo de operación JPEG. En el modo sin pérdidas, las muestras de datos son procesadas una a la vez; por consiguiente, la unidad de datos para los procesos sin perdidas es solo una muestra. Por definición, las muestras son procesadas en las filas de izquierda a derecha y de la fila superior a la fila inferior en cada componente.

Sin embargo, en los modos basados en la DCT, las muestras de la imagen se dividen en bloques de 8x8 pixeles para poder realizar la DCT. Por lo tanto, la unidad de datos para los procesos basados en la DCT es un bloque de 8x8 muestras. Por definición, el bloque superior izquierdo de 8x8 está alineado con el grupo superior izquierdo de 8x8 muestras en cada componente y los bloques en el componente son procesados de izquierda a derecha a través de las filas de bloques, y de la fila de bloques superior a la fila inferior.

Para los procesos de codificación, las unidades de datos son ensambladas en grupos llamados *unidades mínimas de codificación (MCUs)*. Los componentes de color (por ejemplo Y, Cr, Cb) pueden ser procesados por separado (un componente completo a la vez) o en orden entrelazado (por ejemplo un bloque con cada uno de los tres componentes en orden sucesivo). En escaneos con solo un componente, los datos no están entrelazados y la MCU es una unidad de datos. En escaneos con más de un componente sus unidades de datos se agrupan en una sola MCU y están entrelazadas. El número de unidades de datos en una MCU dependerá de la resolución espacial de cada componente.

En la figura 3.6 se muestra un ejemplo de una imagen con sus tres componentes Y, Cr y Cb y la crominancia submuestreada (formato 4:2:2). En las tablas 3.2 y 3.3 se muestra como quedarían las MCU en unidades de datos entrelazadas y no entrelazadas [19].

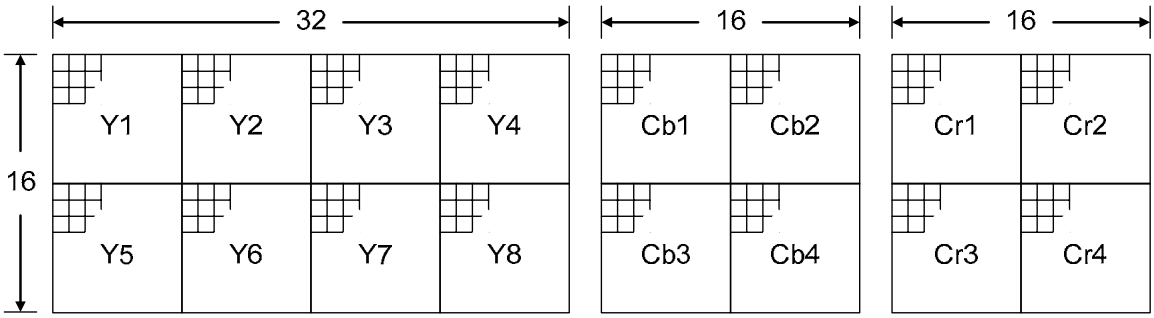


Figura 3.6 Imagen con tres componentes con la crominancia submuestreada (formato 4:2:2).

Componente de Bloque	MCU
Escaneo 1:	
Y1	1
Y2	2
Y3	3
Y4	4
Y5	5
Y6	6
Y7	7
Y8	8
Escaneo 2:	
Cb1	1
Cb2	2
Cb3	3
Cb4	4

Tabla 3.2a Unidades de datos no entrelazadas.

Componente del Bloque	MCU
Escaneo 3:	
Cb1	1
Cb2	2
Cb3	3
Cb4	4

Tabla 3.2b Unidades de datos no entrelazadas.

Componente de Bloque	MCU
Escaneo 1:	
Y1	1
Y2	1
Cb1	1
Cr1	1
Y3	2
Y4	2
Cb2	2
Cr2	2
Y5	3
Y6	3
Cb3	3
Cr3	3
Y7	4
Y8	4
Cb4	4
Cr4	4

Tabla 3.3 Unidades de datos entrelazadas.

El proceso de codificación entrópica siempre se realiza sobre una MCU completa. Por consiguiente, en el codificador cualquier MCU incompleta debe ser codificada copiando la última columna a la derecha y/o la última fila de la parte inferior de cada componente (o por alguna otra técnica de relleno adecuada). Cualquier fila o columna añadida por el codificador no es considerada en la visualización después del proceso de compresión. Por ejemplo, si el ancho del componente de luminancia es de 14 muestras, la segmentación de bloques de 8x8 para la codificación de la DCT requiere dos MCUs horizontales. Las dos columnas que faltan pueden ser obtenidas copiando las muestras de la 14ª columna [1], [19].

3.4.1 Desplazamiento de nivel

Antes de aplicar la DCT a cada bloque de 8x8 muestras de la imagen, los datos de entrada se desplazan en nivel para tener una representación signada restándoles el valor 2^{P-1}, donde *P* es el número de bits de las muestras. Para el caso del modo secuencial básico JPEG en el que P = 8 se tiene un intervalo de 0 a 255, por lo que los datos son desplazados a un intervalo de -128 a 127 [18].

3.4.2 Aplicación de la DCT

En el algoritmo de la DCT en el modo secuencial JPEG, cada bloque de 8x8 es codificado como una unidad completa, independientemente si los datos son entrelazados o no entrelazados. Como resultado de la DCT se obtiene un conjunto de 64 valores llamados coeficientes DCT. Uno de estos valores es el coeficiente DC y los otros 63 son los coeficientes AC. En JPEG los coeficientes DC y AC se codifican de forma distinta, como se explica más adelante [1], [18].

3.4.3 Cuantización

Los 64 coeficientes obtenidos al aplicar la DCT a cada MCU (S_{uv} de la DCT) son cuantizados uniformemente de acuerdo a un tamaño de escalón, el cual está definido por los 64 valores Q_{uv} correspondientes cada elemento S_{uv} en una tabla de cuantización, la cual puede ser definida por el usuario. La ecuación (3.1) define al cuantizador, donde el resultado de la división se redondea al entero más cercano [18].

$$Sq_{uv} = round\left(\frac{S_{uv}}{Q_{uv}}\right) \tag{3.1}$$

donde: $\boldsymbol{S_{uv}}$ es el *valor del coeficiente original*.
$\boldsymbol{Sq_{uv}}$ es el *valor del coeficiente cuantizado*
$\boldsymbol{Q_{uv}}$ es el *tamaño de escalón de cuantización*.

Un valor de Q_{uv} muy grande genera una compresión más alta (debido a que se tiene un mayor número de coeficientes igual a cero después de la cuantización), pero con el costo de tener una distorsión mayor al decodificar la imagen [1].

3.4.4 Codificación Entrópica

Una vez que los coeficientes de la DCT han sido cuantizados, se procede a la codificación entrópica, en la cual se puede utilizar la *codificación Huffman* o bien la *codificación aritmética*. Si se utiliza la codificación Huffman, se deben proporcionar al codificador las tablas de codificación Huffman. No existen valores por default para las tablas Huffman por lo que el usuario puede utilizar tablas adaptadas especialmente al tipo de imágenes que desea comprimir. Si se usa la codificación aritmética, se deben incluir en el codificador las tablas de acondicionamiento, o de lo contrario se pueden usar las tablas que maneja el estándar por default. Los procedimientos y tablas utilizados para codificar los coeficientes AC y DC son un poco distintos, por lo que se procesan de forma separada, como se describe a continuación [1], [4].

Codificación de los coeficientes DC

El coeficiente DC en un bloque de 8x8 pixeles, representa el valor promedio de los 64 pixeles que constituyen la unidad de datos y está en la posición (1,1) del bloque MCU, mientras que la mayoría de los coeficientes AC son valores muy pequeños con respecto al valor del coeficiente DC. La experiencia muestra que en una imagen de tono continuo, los coeficientes DC de las unidades adyacentes no difieren mucho. Para explotar la correlación entre los coeficientes DC de los bloques adyacentes y codificarlos eficientemente se utiliza la codificación DPCM, como se muestra en la figura 3.7 [11], [13].

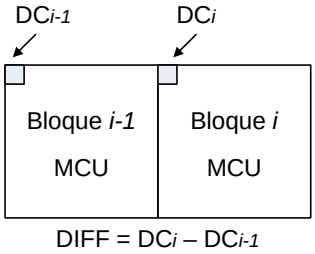


Figura 3.7 Codificación diferencial de los coeficientes DC.

El valor que se codifica es la diferencia DPCM (*DIFF*) entre el coeficiente DC cuantizado del bloque actual DC_i y el coeficiente DC_{i-1} del bloque previo, dentro del mismo componente, como se muestra en la ecuación (3.2).

$$DIFF = DC_i - DC_{i-1} \tag{3.2}$$

El modelo estadístico Huffman para la codificación de diferencias DPCM, agrupa los valores de las diferencias dentro de un conjunto de categorías, cuyas magnitudes se incrementan de forma cuasi logarítmica. Cada una de estas categorías de diferencias representa un símbolo al cual se le asigna un código Huffman (también llamado *código base*). El modelo, aunque no es lo bastante óptimo en términos de la entropía, tiene un alfabeto de símbolos muy pequeño. En la tabla 3.4 se muestran algunos códigos Huffman para los valores de las diferencias DPCM correspondientes a la luminancia, con una precisión de 8 bits [19].

Categoría SSSS	**Diferencia DPCM (DIFF)**		**Código Huffman**
0	0		00
1	-1,	1	010
2	-3, -2,	2, 3	011
3	-7, ..., -4,	4, ..., 7	100
4	-15, ..., -8,	8, ..., 15	101
5	-31, ..., -16,	16, ..., 31	110
6	-63, ..., -32,	32, ..., 63	1110
7	-127, ..., -64,	64, ..., 127	11110
8	-255, ..., -128,	128, ..., 255	111110
9	-511, ..., -256,	256, ..., 511	1111110

Tabla 3.4 Clasificación de las diferencias DPCM en categorías SSSS y sus códigos Huffman.

Posteriormente, una vez que se ha clasificado la diferencia DPCM y se ha identificado el código Huffman que le corresponde, a dicho código se le concatenan *SSSS bits adicionales* para identificar el signo y la magnitud de la diferencia [19].

Las reglas que hay que seguir para concatenar los bits adicionales al valor del código Huffman son las siguientes:

- Si la diferencia DPCM es positiva, se concatenan los *SSSS bits* menos significativos del valor de la diferencia.

- Si la diferencia DPCM es negativa, se concatenan los *SSSS bits* menos significativos del valor de la diferencia pero en complemento a uno.

Los *SSSS bits adicionales* se concatenan al primer bit menos significativo del código Huffman que le corresponde. La tabla 3.5 muestra las secuencias de bits adicionales para unas cuantas categorías de diferencias. Nótese que el primero de los bits adicionales es 1 si la diferencia es positiva, y 0 si la diferencia es negativa [19].

Cat. SSSS	**Diferencia DPCM (DIFF)**	**SSSS 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

Tabla 3.5 Codificación Huffman de las diferencias DPCM.

La tabla 3.5 tiene suficientes entradas para codificar diferencias con cualquier precisión que se requiera en algún modo de operación JPEG. La categoría cero no es usada para símbolos; esta es utilizada para definir el código de fin de bloque (EOB). En la tabla 3.6 se muestran algunos ejemplos de codificación entrópica Huffman JPEG para diferencias DPCM [11], [19].

Valor coeficiente DC cuantizado	+8	+9	+8	-6	-8	-3	+3	+3
Diferencia DPCM	0	+1	-1	-14	-2	+5	+6	0
SSSS	0	1	1	4	2	3	3	0
Bits adicionales	-	1	0	0001	01	101	110	-

Tabla 3.6 Ejemplos de codificación Huffman para diferencias DPCM.

Las tablas de codificación Huffman utilizadas para la codificación de la luminancia y crominancia de los coeficientes DC se muestran en las *tablas A.1* y *A.2* del *anexo A*, respectivamente. Estas dos tablas han sido desarrolladas a través de promedios estadísticos de un gran conjunto de imágenes con precisión de 8 bits [13].

Codificación de los coeficientes AC

A diferencia de la codificación de coeficientes DC, los coeficientes AC son ordenados en zigzag de tal forma que las frecuencias más bajas de la DCT quedan agrupadas al inicio del arreglo. De acuerdo al orden de escaneo zigzag, los coeficientes se pueden representar por la ecuación (3.3).

$$ZZ(0) = S_{q00}, ZZ(1) = S_{q01}, ZZ(2) = S_{q10}, \ldots., ZZ(63) = S_{q77} \quad (3.3)$$

El primer coeficiente del ordenamiento en zigzag es el coeficiente DC y los restantes son los coeficientes AC. El orden de este arreglo se muestra en la figura 3.8 [13], [19].

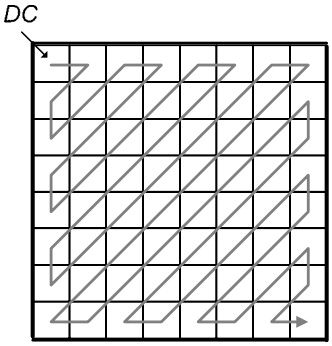


Figura 3.8 Ordenamiento zigzag para los coeficientes de la DCT.

Puesto que muchos de los coeficientes AC se vuelven cero después de la cuantización, el codificador Huffman utiliza símbolos que combinan las *longitudes de ceros* (el número de ceros que preceden a un coeficiente diferente de cero en la secuencia zigzag) con las categorías para las magnitudes de los coeficientes diferentes de cero; dichas categorías se incrementan logaritmicamente del mismo modo que las categorías de diferencias DPCM, por lo que la estrategia de codificación es muy similar en ambos casos. La tabla 3.7 muestra las categoría *SSSS* para los coeficientes AC diferentes de cero [13], [19].

Categoría SSSS	**Coeficientes AC**	
0	0	
1	-1,	1
2	-3, -2,	2, 3
3	-7, ..., -4,	4, ..., 7
4	-15, ..., -8,	8, ..., 15
5	-31, ..., -16,	16, ..., 31
6	-63, ..., -32,	32, ..., 63
7	-127, ..., -64,	64, ..., 127
8	-255, ..., -128,	128, ..., 255
9	-511, ..., -256,	256, ..., 511

Tabla 3.7 Clasificación de los coeficientes AC en categorías SSSS.

Por otro lado, el valor *"RRRR"* define la longitud de ceros. Como en el caso de las tablas para codificación de los coeficientes DC, las tablas de codificación de los coeficientes AC se han desarrollado basándose en el promedio estadístico de un gran conjunto de imágenes con precisión de 8 bits. En la tabla 3.8 la intersección de los valores RRRR y SSSS nos proporcionan el valor del código Huffman que representa al coeficiente AC distinto de cero y a la longitud de ceros que le preceden [19].

		SSSS															
													*	*	*	*	*
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
RRRR	**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	**ZRL**	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

***** No son utilizadas en el modo básico
N/A No aplican para el modo secuencial

Tabla 3.8 Codificación Huffman de los coeficientes AC y las longitudes de ceros que les preceden.

Posteriormente, al código Huffman se le concatenan SSSS bits adicionales, para determinar la magnitud y el signo del coeficiente diferente de cero. El formato para los bits adicionales es el mismo que en la codificación de los coeficientes DC. El valor de la categoría SSSS proporciona el número de bits adicionales requeridos para especificar el signo y la amplitud del coeficiente. Los bits adicionales pueden ser los bits SSSS menos significativos de ZZ(k) cuando ZZ(k) es positivo, o los SSSS bits menos significativos de ZZ(k) en complemento a uno cuando ZZ(k) es negativo, donde ZZ(k) es el k-ésimo coeficiente en el arreglo zigzag. Las tablas de codificación Huffman para los coeficientes AC se pueden encontrar en las *tablas A.3* y *A.4* del *anexo A* [13].

En la tabla 3.8, las entradas marcadas como N/A no se aplican en el modo básico secuencial JPEG, pero si son utilizadas en el modo progresivo. La celda donde se la columna SSSS = 0 y la fila RRRR = 0 está reservada para el símbolo *fin de bloque (EOB)*. Un fin de bloque esta diseñado para indicar que el resto de los coeficientes del bloque en el arreglo zigzag son ceros. En el caso poco usual en el cual el coeficiente 63 es diferente de cero, el fin de bloque no se codifica. El símbolo ubicado en la columna SSSS = 0 y la fila RRRR = 15 se define como *longitud sucesiva de ceros (ZRL)*. ZRL es usado para el caso en el cual la longitud de ceros es mayor a 15. ZRL codifica una longitud de 16 ceros; esto puede ser interpretado como una longitud de 15 ceros seguida por un coeficiente de amplitud cero [19].

3.4.5 Inserción de marcadores

Los datos comprimidos de una imagen JPEG siempre contienen dos marcadores aislados que inician y finalizan los datos comprimidos de la imagen. Entre estos dos marcadores hay típicamente dos o mas segmentos marcadores que contienen encabezados de cuadro, encabezados de escaneo, tablas, y diversos parámetros necesarios para la decodificación, y al menos un segmento codificado entropicamente. Existen tres formatos para los datos de compresión JPEG:

1.- El *formato de intercambio para datos de imágenes comprimidas* incluye todas las tablas que son requeridas por el decodificador. Si los datos de las imágenes comprimidas están destinados para uso general se puede utilizar el formato de intercambio.

2.- El *formato abreviado para datos de imágenes comprimidas* puede omitir algunas o todas las tablas necesarias para la decodificación. Estas tablas deben haber sido heredadas de un flujo de datos comprimido previo o deben ser conocidas a través de otros mecanismos (tales como tablas por default definidas para una aplicación particular).

3.- El *formato abreviado para datos de especificación de tablas* no contiene cuadros, escaneos, o segmentos codificados entropicamente. Este formato abreviado es utilizado para establecer tablas que puedan ser heredadas por el formato abreviado para datos de imágenes comprimidas. Si se usan estos formatos abreviados, la aplicación es la responsable de la coordinación de las especificaciones de tablas y los datos de la imagen comprimida. [19]

Los códigos marcadores son insertados dentro de la secuencia de datos codificados entropicamente. Algunos ejemplos de marcadores son los encabezados de cuadro, los cuales describen los parámetros de la imagen como ancho, alto y número de componentes de color, los encabezados de escaneo y los marcadores de intervalos de reinicio (habilitan un decodificador para re-sincronizarse con la secuencia codificada si ocurre algún error) [1].

Los marcadores tienen una longitud de dos bytes, donde el primer byte siempre tiene el valor hexadecimal 0xFF. El segundo byte contiene el código que especifica el tipo de marcador. Cualquier número de bytes con el valor 0xFF puede ser utilizado como carácter de relleno antes del inicio de algún marcador. El método de compresión JPEG está diseñado para que el valor 0XFF ocurra muy raramente. Cuando se requiere el valor 0XFF en los datos comprimidos, este se codifica con una secuencia de 2 bytes 0XFF seguida por el valor 0X00 para hacer más fácil la exploración de marcadores específicos en un archivo JPEG para las aplicaciones.

El estándar JPEG es bastante flexible cuando se deben de ordenar los marcadores dentro de un archivo. La única regla estricta es que un archivo debe iniciar con un marcador *Inicio de imagen (SOI)* y terminar con un marcador *Fin de imagen (EOI).* A un marcador SOI le siguen los datos comprimidos. Los marcadores JPEG se muestran en la tabla 3.9. En la tabla se puede observar que muchos de los marcadores no son utilizados en el modo básico JPEG [3].

Valor	Símbolo usado en el estándar JPEG	Descripción
FF01*	TEM	Marcador temporal para codificación aritmética
FFD0-FFD7	$RST_0 – RST_7$	Marcador de reinicio
FFD8	SOI	Inicio de Imagen
FFD9	EOI	Fin de Imagen
FFC0	SOF_0	Inicio de cuadro, modo básico
FFC1	SOF_1	Inicio de cuadro, modo secuencial extendido
FFC2*	SOF_2	Inicio de cuadro, modo progresivo
FFC3*	SOF_3	Inicio de cuadro, modo sin pérdidas
FFC4	DHT	Define tablas Huffman
FFC5*	SOF_5	Inicio de cuadro, modo secuencial diferencial
FFC6*	SOF_6	Inicio de cuadro, modo progresivo diferencial
FFC7*	SOF_7	Inicio de cuadro, modo sin pérdidas diferencial
FFC8*	JPG	Reservado
FFC9*	SOF_9	Inicio de cuadro, modo secuencial extendido, codificación aritmética
FFCA*	SOF_{10}	Inicio de cuadro, modo progresivo, codificación aritmética
FFCB*	SOF_{11}	Inicio de cuadro, modo sin pérdidas, codificación aritmética
FFCC*	DAC	Define condiciones de codificación aritmética
FFCD*	SOF_{13}	Inicio de cuadro, modo secuencial diferencial, codificación aritmética
FFCE*	SOF_{14}	Inicio de cuadro, modo progresivo diferencial, codificación aritmética
FFCF*	SOF_{15}	Inicio de cuadro, modo sin pérdidas diferencial, codificación aritmética
FFC0	SOF_0	Inicio de cuadro, modo básico
FFC1	SOF_1	Inicio de cuadro, modo secuencial extendido
FFC2*	SOF_2	Inicio de cuadro, modo progresivo
FFC3*	SOF_3	Inicio de cuadro, modo sin pérdidas
FFC4	DHT	Define tablas Huffman
FFC5*	SOF_5	Inicio de cuadro, modo secuencial diferencial
FFC6*	SOF_6	Inicio de cuadro, modo progresivo diferencial
FFC7*	SOF_7	Inicio de cuadro, modo sin pérdidas diferencial
FFC8*	JPG	Reservado
FFC9*	SOF_9	Inicio de cuadro, modo secuencial extendido, codificación aritmética
FFCA*	SOF_{10}	Inicio de cuadro, modo progresivo, codificación aritmética
FFCB*	SOF_{11}	Inicio de cuadro, modo sin pérdidas, codificación aritmética
FFCC*	DAC	Define condiciones de codificación aritmética
FFCD*	SOF_{13}	Inicio de cuadro, modo secuencial diferencial, codificación aritmética
FFCE*	SOF_{14}	Inicio de cuadro, modo progresivo diferencial, codificación aritmética
FFCF*	SOF_{15}	Inicio de cuadro, modo sin pérdidas diferencial, codificación aritmética
FFDA	SOS	Inicio de escaneo
FFDB	DQT	Define tablas de cuantización
FFDC*	DNL	Define número de líneas
FFDD	DRI	Define intervalo de reinicio
FFDE*	DHP	Define progresión jerárquica
FFDF*	EXP	Componentes de referencia expandida
FFE0-FFEF	APP_0 - APP_{15}	Datos de aplicación específicos
FFFE	COM	Comentario
FFF0-FFFD*	JPG_0 - JPG_{13}	Reservado
FF02-FFBF*	RES	Reservado

*No utilizado en el modo básico JPEG.

Tabla 3.9 Definición de marcadores JPEG.

3.4.6 Descompresión JPEG en modo básico

El resultado de la codificación de imágenes utilizando JPEG es una secuencia de bits que representa los datos de la imagen, y que pueden ser transmitidos o almacenados. Para ver la imagen, esta debe ser decodificada aplicando la descompresión, iniciando con la detección de marcadores, la decodificación entrópica, el proceso de decuantización, la aplicación de la DCT inversa y por último el corrimiento de nivel, como se muestra en la figura 3.9. Debido a que la cuantización es un proceso no reversible, la imagen decodificada no será exactamente igual a la imagen original [1].

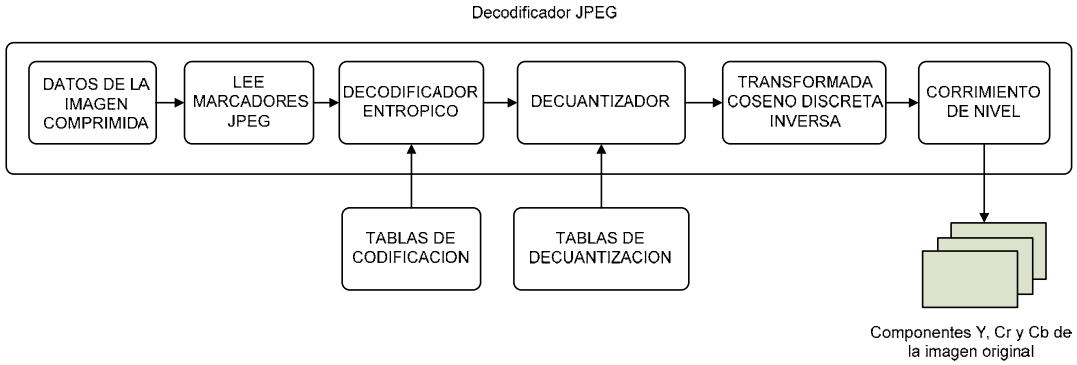


Figura 3.9 Esquema general de descompresión JPEG.

3.5 Esquema de compresión y descompresión MJPEG

Como se mencionó anteriormente, el método de compresión MJPEG consiste en aplicar la compresión JPEG a cada cuadro perteneciente a una secuencia de video. En la figura 3.10 se muestra el diagrama a bloques del proceso de compresión de video MJPEG.

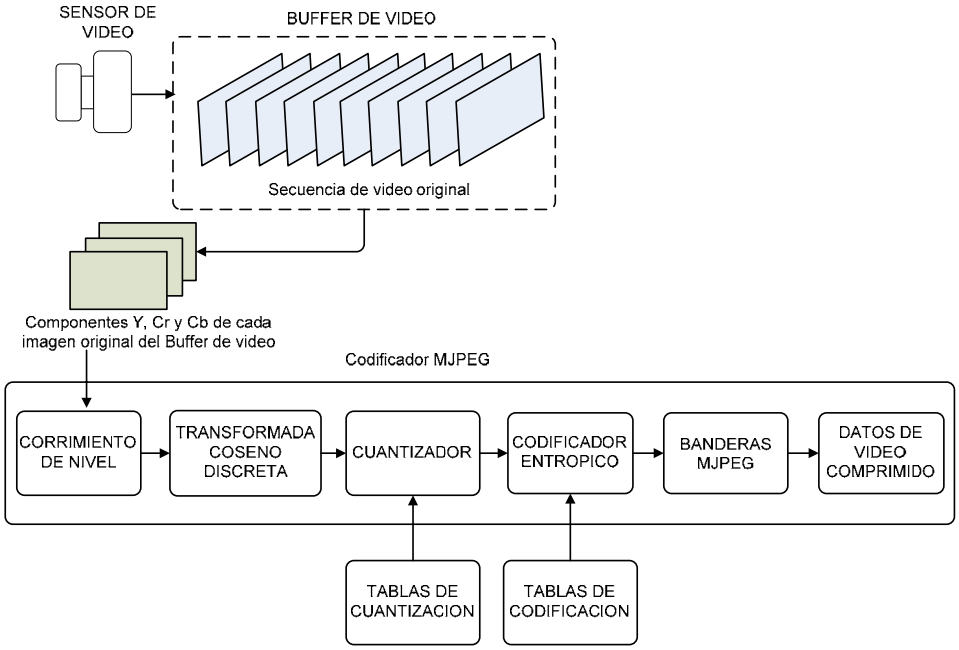


Figura 3.10 Diagrama a bloques de la compresión de video MJPEG.

De la figura 3.10, se puede observar que un sensor de video se encarga de capturar las imágenes que componen una secuencia de video, la cual es almacenada en un buffer de video. Posteriormente, a cada imagen de la secuencia de video se le aplica la compresión JPEG, la cual contiene las etapas de corrimiento de nivel, DCT, cuantización, codificación entrópica y la inserción de marcadores. Las principales banderas que necesita MJPEG son las de inicio de imagen SOI y las de fin de imagen EOI. Una vez que se han insertado las banderas los datos están listos para ser transmitidos o almacenados.

En la figura 3.11 se muestra el diagrama a bloques para la descompresión de video MJPEG. El descompresor primero lee las banderas para detectar el inicio de los datos comprimidos de una imagen. Posteriormente, a los datos comprimidos se les aplica la descompresión JPEG para reconstruir la imagen. Este proceso se repite hasta reconstruir toda la secuencia de video para que pueda ser visualizada posteriormente.

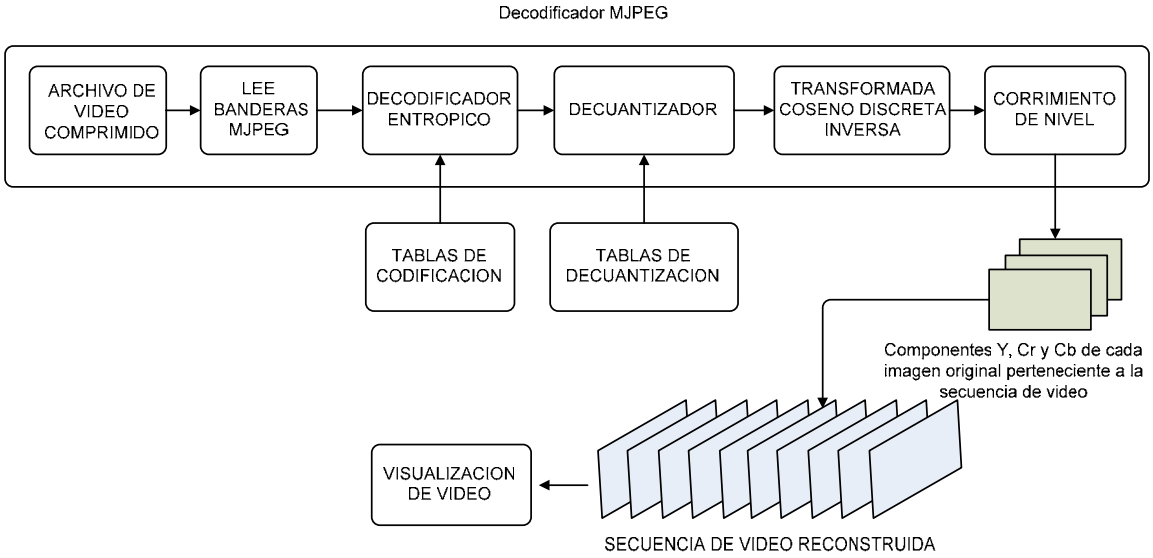


Figura 3.11 Diagrama a bloques de la descompresión de video MJPEG.

RESUMEN

En este capítulo se describió el método de compresión de video MJPEG, así como sus ventajas y desventajas con respecto a otros métodos de compresión de video. Cabe resaltar que MJPEG es el método utilizado para llevar a cabo la compresión de video en este trabajo.

Por otro lado, dado que el método MJPEG está basado en el estándar de compresión de imágenes JPEG, en este capítulo se hizo una descripción de este estándar, teniendo una breve descripción de los diferentes métodos de compresión JPEG existentes. Como en este trabajo se utilizó el modo básico JPEG, se hizo un análisis de las etapas de corrimiento de nivel, aplicación de la Transformada Coseno Discreta, proceso de cuantización, codificación entrópica e inserción de marcadores, tomando como referencia el modo básico JPEG. Además, se realizó una breve descripción del proceso de descompresión JPEG, y del esquema de compresión y descompresión MJPEG. En el siguiente capítulo se desarrolla la implementación hardware-software del diagrama de la figura 3.10, y con la finalidad de evaluar el desempeño del codificador se implementa el proceso de descompresión de la figura 3.11.

4.- SISTEMA DE ADQUISICION Y TRANSFERENCIA DE VIDEO AL DSP C6416

Este capítulo tiene como objetivo presentar el desarrollo e implementación del sistema de adquisición de video, el DSPC6416 y la tarjeta de desarrollo a utilizar. Posteriormente, se analiza el hardware utilizado para la captura de video y se hace la descripción de las diferentes etapas necesarias para implementar la adquisición de video, desde la configuración e inicialización del sensor de video, hasta la transferencia de los datos de video utilizando un canal de Acceso Directo a Memoria Mejorado (EDMA), el uso de un triple buffer y la adaptación de las imágenes de video de un formato QVGA 4:2:2 a un formato QCIF 4:2:0.

4.1 Descripción general del sistema

Para llevar a cabo la adquisición y compresión de video MJPEG se necesita un sensor de video, al igual que un hardware de alto desempeño para realizar los procesos del sistema de forma eficiente. Cuando se requiere que los procesos se ejecuten en tiempo real, es necesario realizarlos dentro del tiempo de muestreo de la señal de video. *Tiempo real* significa que el procesamiento se debe completar dentro del tiempo disponible entre dos muestras consecutivas, el cual depende de la aplicación, como se observa en la figura 4.1. En nuestro caso, la tasa de cuadros por segundo es un mínimo de 10 fps, la cual es bastante aceptable para transmisión de video. Si consideramos un tamaño de video QVGA de 320x240 a 10 fps, el tiempo de muestreo de la señal de video será de 1.3 μseg [26].

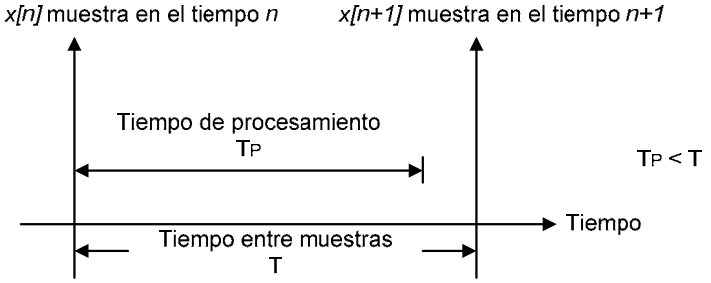


Figura 4.1 Esquema de tiempo disponible entre muestras para ejecutar el proceso en tiempo real.

Por otro lado, lograr la aproximación al tiempo real tiene sus dificultades, ya que se requieren optimizaciones en la parte algorítmica y el software, además de contar con arquitecturas de procesamiento de alto desempeño. Los *procesadores digitales de señales (DSPs)* son microprocesadores de propósito especial con una arquitectura y un conjunto de instrucciones diseñados especialmente para el procesamiento de señales, principalmente para aplicaciones en tiempo real. La familia de procesadores digitales de señales TMS320C6000 de Texas Instruments es adecuada para realizar cálculos numéricos intensivos y es muy utilizada para el procesamiento de imágenes y video [27].

La implementación del sistema de adquisición y compresión de video MJPEG propuesto consta de las etapas que se muestran en la figura 4.2:

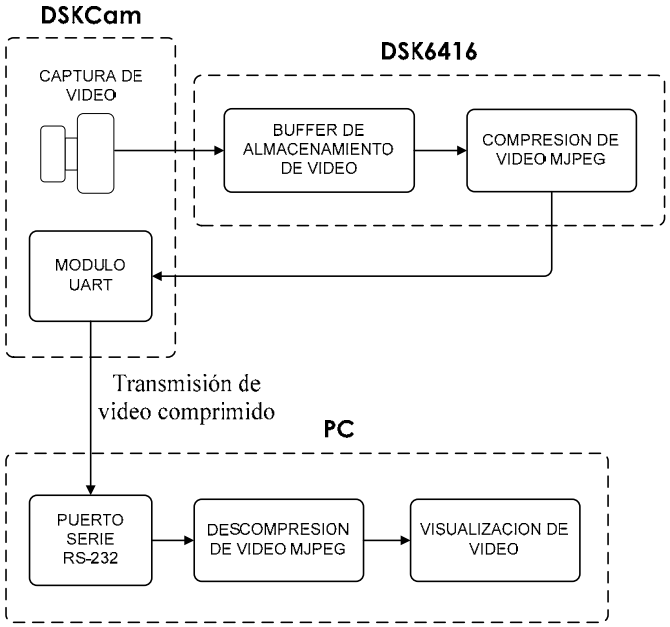


Figura 4.2 Diagrama a bloques del sistema de adquisición y compresión de video MJPEG.

- *Adquisición de video:* Se lleva a cabo utilizando un sensor de video, el cual se encarga de adquirir una secuencia de video en formato digital y almacenarla en un buffer de video.

- *Compresión de video MJPEG:* En esta etapa, se aplica el método de compresión MJPEG, el cual consiste en comprimir cada imagen almacenada en el buffer de video mediante el estándar JPEG, utilizando el DSPC6416.

- *Transmisión*: El video comprimido es enviado a la PC a través del puerto serie RS-232 (Esta etapa esta fuera del alcance del proyecto y esta planeada a desarrollarse a futuro).

- *Descompresión MJPEG:* Esta etapa es inversa al proceso de compresión y su finalidad es reconstruir el video original a partir de los datos de video comprimido utilizando la PC.

- *Visualización de Video:* El video recuperado es desplegado en el monitor de la PC.

4.2 Procesador Digital de Señales DSP TMS320C6416

Las arquitecturas de los procesadores digitales de señales (DSP) actuales proporcionan una solución conveniente que combina flexibilidad y potencia computacional. El diseño de la familia de DSPs TMS320C6xxx (C6xxx) de la marca Texas Instruments está orientada al procesamiento digital de imágenes. En este trabajo, se decidió utilizar el DSP C6416 de punto fijo de alto desempeño (versión de 1Ghz), el cual está basado en la tecnología VelociTI avanzado (velociTI.2), con arquitectura de *palabra de instrucción muy larga (VLIW)* desarrollado por Texas Instruments, en la cual varias palabras de datos son capturadas y procesadas simultáneamente. La tecnología VelociTI es una modificación de la arquitectura VLWI que reduce el tamaño de código e incrementa el desempeño cuando las instrucciones están almacenadas en memoria externa al chip [10], [26].

El DSP C6416 alcanza un desempeño de 8000 *Millones de Instrucciones por Segundo (MIPS)* con una frecuencia de reloj de 1 GHz, por lo que su ciclo de instrucción es de 1ns; además, puede ejecutar ocho instrucciones de 32 bits/ciclo y 28 operaciones/ciclo. El núcleo del DSP consta de 64 registros de propósito general con una longitud de palabra de 32 bits y ocho unidades funcionales independientes de 2 multiplicadores para un resultado de 32 bits, además de 6 *Unidades Lógicas Aritméticas (ALUs)*. El DSP C6416 puede procesar 4 *Multiplicaciones – Acumulaciones (MACs)* de 16 bits por ciclo para un total de 4000 *millones de MACs por segundo (MMACS)*, o bien 8 MACS de 8 bits por ciclo para un total de 8000 MMACS.

Entre los periféricos que contiene el DSP están tres *Puertos Seriales Buffereados Multicanal (McBSPs)*, tres timers de 32 bits de propósito general, una *Interface al Puerto Huésped (HPI)* y dos *Interfaces de Memoria Extendida (EMIF)*: la *EMIFA* de 64 bits y la *EMIFB* de 16 bits. Además, el DSP C6416 cuenta con 64 canales de *Acceso Directo a Memoria Mejorado (EDMA)* independientes. El uso del EDMA es muy importante en este proyecto, ya que una vez iniciado, realiza transferencias de datos sin la intervención del CPU [10].

4.2.1 Acceso Directo a Memoria Mejorado (EDMA)

El acceso directo a memoria mejorado permite la transferencia de datos de una sección de memoria a otra (memoria interna, interface de memoria externa o periféricos) sin la intervención del CPU. Los 64 canales de EDMA pueden ser configurados independientemente para transferencias de datos. El canal EDMA puede transferir palabras de datos de 8, 16 y 32 bits y es configurado a través de un conjunto de registros como: dirección (fuente y destino), índices, tamaño de palabra, número de elementos, número de frames, tipo de transferencia y registros de control. Las direcciones de la fuente y el destino pueden ser direcciones de memoria de programa interna, memoria de datos interna, de una interface de memoria externa o de un bus de periféricos internos. La frecuencia de reloj del EDMA del DSP C6416 es de ½ de la frecuencia de reloj del CPU (CLK/2).

Existen tres formas para iniciar una transferencia de datos por EDMA:

- *Solicitud de transferencia por evento:* Permite que dado un evento generado por un periférico, el sistema, o algún evento externo comience la transferencia EDMA.
- *Solicitud de transferencia por encadenamiento:* Cuando un canal EDMA termina su transferencia, inicia la transferencia de datos de otro canal EDMA.
- *Solicitud de transferencia por CPU:* El CPU es el que inicia la transferencia EDMA.

Cada canal EDMA puede ser programado con una prioridad, considerando que el canal 0 es el que tiene la más alta prioridad, además, los canales EDMA pueden ser configurados para comenzar una transferencia de datos de forma independiente. El EDMA proporciona dos tipos de transferencias de datos, unidimensional (1D) y bidimensional (2D). El número de dimensiones en una transferencia determinará la composición de un *frame* de datos. En la transferencia 1D, el *frame* está constituido por un número determinado de elementos individuales. En una transferencia 2D, los bloques están formados por un número de vectores, cada uno de los cuales está formado por un determinado número de elementos [23], [27].

Las transferencias 2D son muy utilizadas en aplicaciones de imágenes y video donde un conjunto de elementos contiguos (referidos como vectores) tienen que ser transferidos cuando se recibe un evento de sincronía. En este tipo de transferencias no hay espacios o índices entre los elementos de un vector. El número de elementos en un vector establece la primera dimensión de la transferencia, la segunda dimensión está determinada por un grupo de vectores, llamado bloque. Los vectores pueden estar separados uno del otro por una cantidad determinada. La figura 4.3 muestra el diagrama de una transferencia en 2D así como la composición de un bloque de dos dimensiones, con un contador de vectores *n* y un contador de elementos *m* [23], [27].

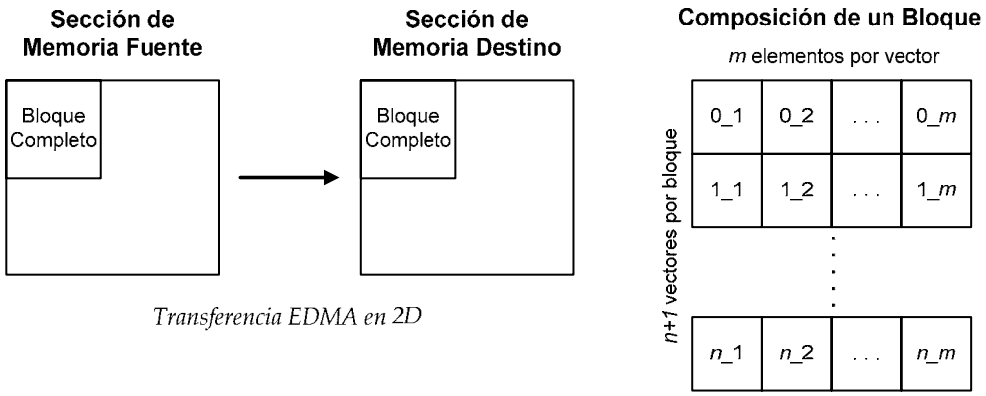


Figura 4.3 Transferencia EDMA bidimensional y composición de un bloque.

4.2.2 Procesamiento de Interrupciones

Como su nombre lo indica, una *interrupción* ocasiona que el DSP detenga cualquier proceso que esté realizando para ejecutar una tarea ligada con la interrupción, la cual se conoce como *Rutina de Servicio de Interrupción (ISR)*. Cuando ocurre una interrupción, el flujo del programa es re-direccionado a la ISR de la interrupción. Al momento de la interrupción, las condiciones del proceso que se estaba ejecutando deben ser salvadas, con la finalidad de ser restauradas después de que termina de ejecutarse la rutina ISR, esto quiere decir que al llegar la interrupción, se salvan los registros y se pasa a procesar la ISR.

Las interrupciones son muy utilizadas para el procesamiento de datos en tiempo real, debido a que elimina la necesidad de esquemas complicados de sincronización. Una interrupción puede ser generada interna o externamente, y la fuente de interrupción puede ser un convertidor analógico digital, un timer o un pin externo.

Existen dieciséis fuentes de interrupción para el DSP C6416, las cuales incluyen dos interrupciones para timers, cuatro interrupciones externas, cuatro para los puertos seriales McBSP, una interrupción para los canales EDMA, cuatro de propósito especial y dos reservadas. De las interrupciones INT4 – a la INT11 la prioridad más alta la tiene la interrupción 4. La selección de la interrupción se realiza a través de un *selector de interrupciones* [26], [27].

4.2.3 Tarjeta de desarrollo DSKC6416T

La tarjeta de desarrollo DSK6416 es una plataforma de bajo costo que permite al usuario evaluar y desarrollar aplicaciones sobre el DSP C6416. Los principales componentes de la tarjeta utilizados en este proyecto se muestran en la figura 4.4 y son:

- Un procesador digital de señales DSP TMS320C6416T, el cual opera a 1 Ghz.
- 16 Mbytes de memoria externa SDRAM.
- Conectores de expansión estándar para el uso de una tarjeta externa.

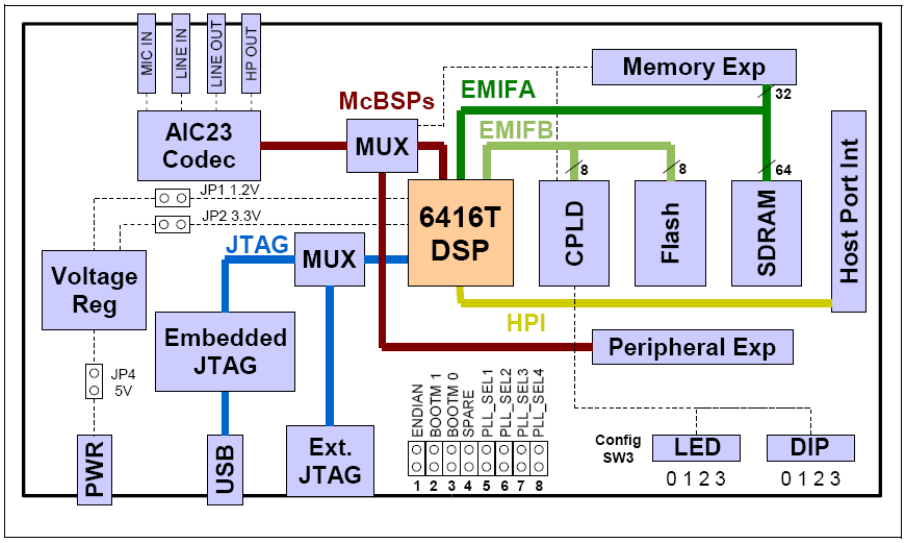


Figura 4.4 Diagrama a bloques de la tarjeta de desarrollo DSKC6416T.

Uno de los factores por lo que se decidió utilizar la tarjeta DSK6416 es que contiene 16 Mbytes de memoria externa SDRAM, lo cual nos permite almacenar aproximadamente 25 cuadros de video en tamaño VGA a color con submuestreo 4:2:2, 100 cuadros a color en formato QVGA 4:2:2 y 400 cuadros en tamaño QCIF a color con submuestreo 4:2:0, lo cual es suficiente para almacenar una secuencia de video y aplicarle el proceso de compresión MJPEG, con el fin de evaluar tiempos de procesamiento. Además, la tarjeta DSK6416 cuenta con conectores de expansión para la comunicación con una tarjeta externa.

Funcionamiento de la tarjeta de desarrollo DSKC6416

En la figura 4.4 se muestra como están conectados los diferentes dispositivos de la tarjeta DSK6416. El DSP C6416 se comunica con los periféricos instalados en la tarjeta a través de alguno de los dos buses de la Interface de Memoria Extendida: el *EMIFA* con un ancho de 64 bits y el *EMIFB* con un ancho de 8 bits. Las memorias SDRAM y Flash están conectadas a uno de estos buses. La EMIFA también está enlazada a los conectores de la tarjeta de expansión. La tarjeta utiliza un *Dispositivo Lógico Programable Complejo (CPLD)* para establecer la comunicación entre los diferentes componentes de la tarjeta. A su vez, el CPLD tiene un registro de interface de usuario que le permite configurar la tarjeta para realizar funciones de lectura y escritura a los registros del CPLD [21].

Para programar el DSP e interactuar con la tarjeta de desarrollo se utiliza el ambiente integral *Code Composer Studio (CCS)* de Texas Instruments, el cual se comunica con la tarjeta DSK6416 a través de un emulador JTAG con una interface USB. La tarjeta DSK6416 tiene 8 switchs de configuración para controlar el modo de operación del DSP. Por default la tarjeta inicia desde el EMIFB en modo *'big endian'* (*sección 3.2*) con una frecuencia de reloj del CPU de 1 GHz y 125 MHz de frecuencia en la EMIFA [21].

4.2.4 Sincronización con Semáforos en el DSPC6416

El software de desarrollo Code Composer Studio proporciona un sistema operativo reconfigurable llamado DSP/BIOS para el análisis, programación e intercambio de datos en tiempo real. El DSP/BIOS proporciona un conjunto de funciones fundamentales para la sincronización y comunicación entre tareas, a través de *semáforos*. Los semáforos se utilizan frecuentemente para coordinar accesos a recursos compartidos entre un conjunto de tareas competentes. Los *objetos SEM* son contadores de semáforo que pueden ser utilizados para la sincronización de tareas o para exclusión mutua. Los contadores de semáforo mantienen un conteo interno del número de recursos asociados que están disponibles.

Dentro de las funciones de los objetos SEM se tiene la función *SEM_pend( )* la cual pone en estado de espera al proceso o tarea asociado al semáforo. El parámetro *timeout* de la función SEM_pend configura a la tarea/proceso para *esperar un tiempo determinado, esperar indefinidamente (SYS_FOREVER), o no esperar nada (0)*. La función *SEM_post( )* es utilizada para liberar un semáforo, es decir, habilita la tarea/proceso para que sea procesada por el CPU. La función *SEM_ipost* es similar a la función SEM_post( ), solo que SEM_post es usada dentro de una tarea y SEM_ipost es llamada dentro de una rutina ISR [27], [28], [29].

4.3 Tarjeta de expansión DSKcam

Para la realizar la adquisición de video se decidió utilizar la tarjeta de expansión *DSKCam* la cual utiliza un sensor de video tipo CMOS que es compatible con la tarjeta DSK6416. La ventaja de los sensores CMOS es que son de bajo costo comparados con los sensores CCD, requieren bajo consumo de potencia, son de tamaño pequeño y tienen una resolución de imagen media. Además, los sensores de video CMOS integran las partes analógicas y digitales del circuito en un solo chip, proporcionando una interface directa al bus de datos del procesador, con lo cual se reduce significativamente el número de componentes y el consumo de potencia.

En la figura 4.5 se muestra el diagrama a bloques de la tarjeta de expansión DSKcam. Esta tarjeta utiliza específicamente un sensor de video CMOS Omnivision OV7620. El buffer de video está formado por dos buffers de memoria del tipo *Primero en Entrar Primero en Salir (FIFO)*. Los buffers FIFO tienen la capacidad suficiente para almacenar un cuadro a color con resolución VGA. El tamaño del buffer de la DSKcam para los cuadros de video es de 6 Mbits. Un dispositivo CPLD controla el buffer FIFO de la DSKcam y las funciones lógicas de la tarjeta, a través de registros mapeados, como se describe más adelante. La tarjeta DSKcam incluye un módulo de red Ethernet TCP/IP embebido, basado en el dispositivo WizNET W3100A y un módulo *Transmisor Receptor Asíncrono Universal (UART)* 16C550 con 9 hilos, tipo D, compatible con el protocolo de transmisión serial RS232 [22].

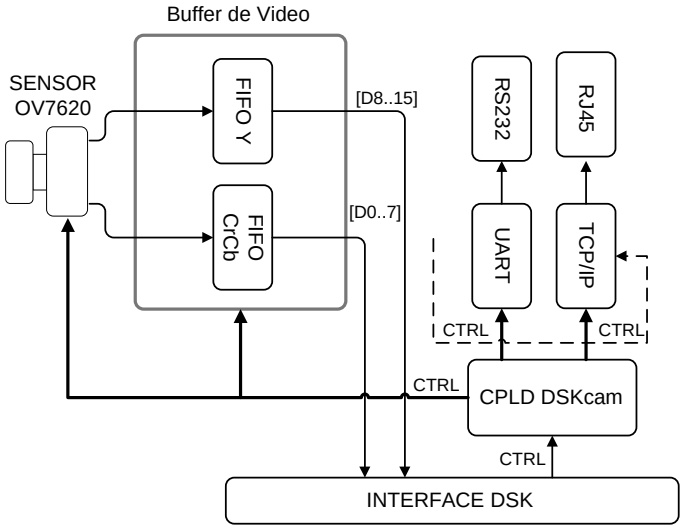


Figura 4.5 Diagrama a bloques de la tarjeta DSKcam.

4.4 Implementación del sistema de adquisición de video a nivel hardware

En la figura 4.6 se muestra el esquema general del sistema de adquisición de video a nivel hardware. La adquisición de video funciona de la siguiente forma:

- Antes de empezar a adquirir imágenes, el sensor de video debe ser iniciado y configurado a través del bus I2C. Esto lo hace el DSP escribiendo a los registros de control I2C mapeados en memoria, como se observa en la figura 4.6. Este proceso se describe más adelante.

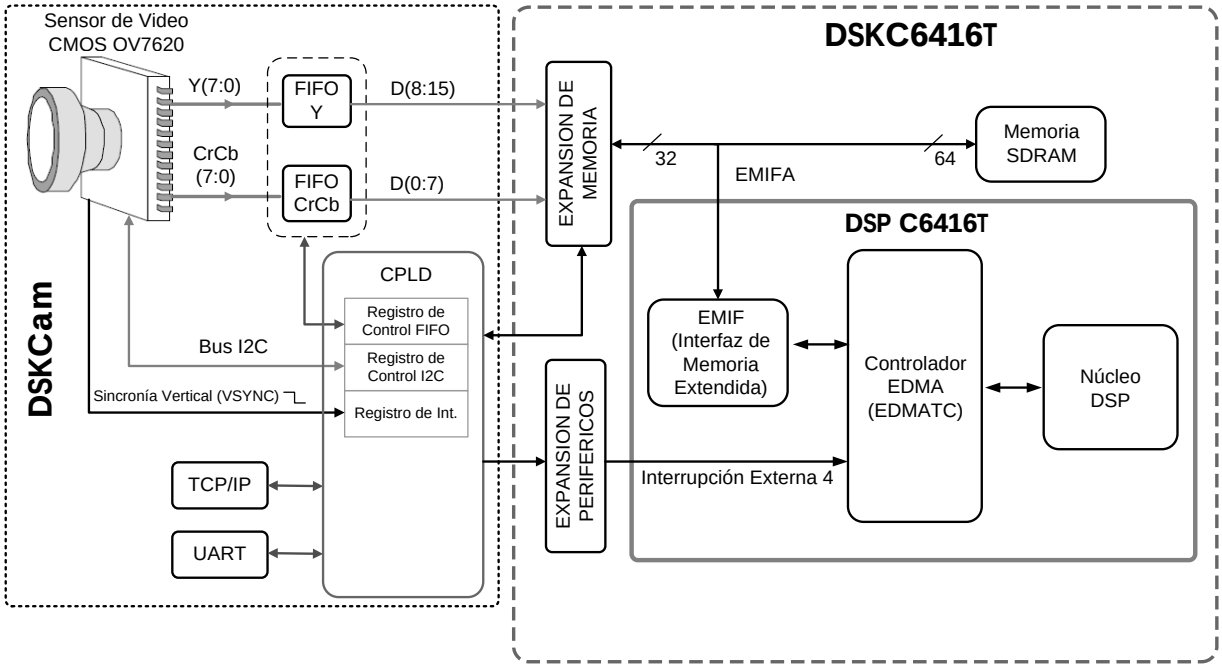


Figura 4.6 Diagrama general del hardware del sistema de adquisición de video.

- Una vez que el sensor de video está inicializado y configurado, éste empieza a adquirir imágenes de video a una tasa de 30 fps, en tamaño QVGA con formato 4:2:2. Los datos de las imágenes son transferidos a los buffers de memoria FIFO Y y FIFO CrCb, como se muestra en la figura 4.6. Los 8 bits más significativos corresponden al componente Y, y los 8 menos significativos corresponden a los componentes Cr y Cb. Para evitar un desbordamiento de cuadros, la lectura y escritura a los buffers FIFO se controla a través de los registros de control del FIFO mapeados en memoria.

- Cuando el sensor de video termina de adquirir una imagen completa, genera un pulso de *sincronía vertical (VSYNC)*, el cual genera una interrupción por hardware al DSP, a través de la interrupción externa 4.

- Al momento de recibir la interrupción, el DSP inicia una transferencia de datos por EDMA moviendo los datos de las imágenes de video desde los buffers de memoria FIFO a una sección de memoria SDRAM de la tarjeta DSKC6416, para que posteriormente sean procesadas por el DSP.

En las siguientes secciones se describen las etapas de la adquisición de video.

4.4.1 Funcionamiento del sensor de video

El sensor de video CMOS OV7620 soporta imágenes con resolución VGA, y contiene un procesador de señal analógico, convertidores AD de 10 bits, un formateador de datos digitales, un puerto de video y registros de control con una interface del *Bus Serial de Control de la Cámara (SCCB)*. Los registros de control incluyen generación de tiempo, bloque de exposición y balance de blanco. Estos bloques se muestran en la figura 4.7 [25].

En la figura 4.7 se observa que una vez que se obtiene una imagen a través de la matriz de elementos de sensado, el bloque de procesamiento analógico ejecuta la mayor parte del procesamiento de la señal, haciendo la separación de color, AGC (Control Automático de Ganancia), corrección gamma, corrección de color, balance de color, calibración del nivel de negro, corrección de apertura, controles para la luminancia y crominancia de la imagen, y filtrado anti-aliasing. Todas estas funciones son programadas a través de los registros de control del sensor. El sensor de video OV7620 utiliza el bus serial *SCCB*, el cual está basado en el protocolo *Circuito Inter-Integrado (I^2C)* de 2 hilos, para programar los registros internos de control.

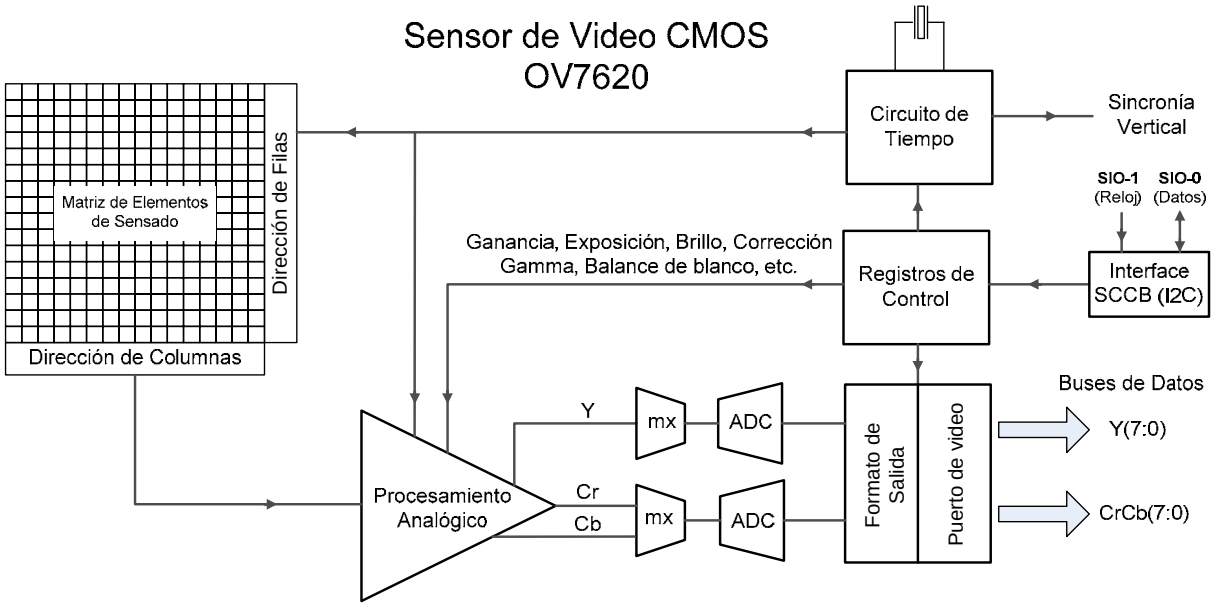


Figura 4.7 Diagrama a bloques del sensor de video CMOS OV7620.

El sensor de video se puede programar para manejar imágenes con resolución VGA o QVGA, y espacio de color RGB o YCrCb con submuestreo 4:2:2. La tasa de cuadros por segundo capturados por el sensor puede ser de 30 fps o menor, con una resolución de video VGA, o bien se puede tomar una solo imagen mediante un solo disparo. En nuestro caso, donde deseamos realizar la compresión MJPEG, lo más conveniente es utilizar resolución QVGA (aunque después se pasa a una resolución QCIF mediante el software que diseñamos como se describe más adelante) y el espacio YCrCb con submuestreo 4:2:2, debido a que esto reduce la cantidad de datos a procesar (*sección 1.5.2*).

La señal YCrCb es alimentada a dos convertidores A/D de 10 bits, que operan arriba de 13.5 MSPS, uno para el canal Y y el otro es compartido por los canales Cr y Cb. Posteriormente, el flujo de datos ya convertidos es acondicionado en el formateador digital. Finalmente los datos de 16 bits son multiplexados al puerto de video digital, (8 bits son para la componente de luminancia y 8 bits para la crominancia). Este puerto, como se observa en la figura 4.6, está conectado a 2 buffers de memoria FIFO, uno para los datos de luminancia y otro para las crominancias. En la tabla 4.1 se muestra la secuencia de 16 bits que el sensor de video CMOS transfiere a los FIFOs a través del puerto digital, con un formato de imagen YCrCb con submuestreo 4:2:2.

Como se mencionó en la *sección 1.5.2*, cuando se utiliza el submuestreo 4:2:2 en una imagen, por cada 2 pixeles en la dirección horizontal, se toman las 2 muestras de luminancia de ambos pixeles y únicamente se toman las muestras de crominancia del primer pixel. En la tabla 4.1 se puede observar que en el bus de datos Y del puerto de video digital ($b_{15} - b_8$), el primer byte, el cual está en la columna del tiempo t_0, corresponde al componente de luminancia del primer pixel, mientras que en el bus de datos CrCb ($b_7 - b_0$) el primer byte en t_0 pertenece al componente de crominancia azul Cb del primer pixel. El segundo byte en el bus de datos Y en el tiempo t_1 corresponde al componente de luminancia del segundo pixel, mientras que el byte del bus de datos CrCb en t_1 es el de la crominancia roja Cr del primer pixel. Las dos últimas filas de la tabla 4.1 indican a que pixel pertenecen los bytes de cada componente de color.

	Bus de datos	Secuencia de los bytes de los pixeles					
		t_0	t_1	t_2	t_3	t_4	t_5
Bus de datos Y	b_{15}	Y7	Y7	Y7	Y7	Y7	Y7
	b_{14}	Y6	Y6	Y6	Y6	Y6	Y6
	b_{13}	Y5	Y5	Y5	Y5	Y5	Y5
	b_{12}	Y4	Y4	Y4	Y4	Y4	Y4
	b_{11}	Y3	Y3	Y3	Y3	Y3	Y3
	b_{10}	Y2	Y2	Y2	Y2	Y2	Y2
	b_9	Y1	Y1	Y1	Y1	Y1	Y1
	b_8	Y0	Y0	Y0	Y0	Y0	Y0
							→ t
Bus de datos CrCb	b_7	Cb7	Cr7	Cb7	Cr7	Cb7	Cr7
	b_6	Cb6	Cr6	Cb6	Cr6	Cb6	Cr6
	b_5	Cb5	Cr5	Cb5	Cr5	Cb5	Cr5
	b_4	Cb4	Cr4	Cb4	Cr4	Cb4	Cr4
	b_3	Cb3	Cr3	Cb3	Cr3	Cb3	Cr3
	b_2	Cb2	Cr2	Cb2	Cr2	Cb2	Cr2
	b_1	Cb1	Cr1	Cb1	Cr1	Cb1	Cr1
	b_0	Cb0	Cr0	Cb0	Cr0	Cb0	Cr0
	Componente	**Pixel al que pertenece**					
	Y	0	1	2	3	4	5
	CbCr	0		2		4	

Tabla 4.1 Orden de los 16 bits en el puerto de video digital del sensor (formato YCrCb 4:2:2).

Finalmente, cuando el sensor de video termina de transferir una imagen completa a los buffers FIFO, genera un pulso de *sincronía vertical (VSYNC)*, escribiendo al registro de interrupciones del CPLD, con la finalidad de causar una interrupción por hardware al DSP y que los datos de la imagen sean transferidos a la memoria SDRAM de la tarjeta DSK6416 como se mostró en la figura 4.6 [25].

4.4.2 Configuración e inicialización de la cámara

Antes de iniciar la captura de video, los registros del sensor de video deben ser configurados. Para ello, el sensor de video CMOS OV7620 cuenta con 124 registros, los cuales se configuran a través del bus serial SCCB. Para configurar la cámara, el DSP implementa la comunicación I2C con el sensor de video escribiendo al registro de control I2C del CPLD de la DSKcam (el cual está mapeado en memoria en la dirección 0xA0020000) y se describe en la tabla 4.2.

Posición del Bit	R/W	Función
D0 (I2CDATA)	R/W	Esta señal lee y controla la línea de datos I2C
D1 (I2CLK)	R/W	Esta señal maneja la línea del reloj I2C
D2 (Sin Uso)	-	
D3 (Sin Uso)	-	

Tabla 4.2 Registro de Control I2C.

El bus SCCB está compuesto por la línea *SIO-1, que es utilizada para la señal de reloj I2C* y por la línea *SIO-0, la cual es la línea de datos I2C*. El sensor de video opera como dispositivo esclavo. A continuación se describe el proceso de comunicación a través del bus SCCB [24].

Transmisión de datos por el bus SCCB

El proceso de escritura de datos al dispositivo esclavo (el sensor de video) se define como transmisión de escritura, mientras que la lectura de datos del dispositivo esclavo se define como transmisión de lectura. Para empezar la transmisión de datos, el dispositivo maestro (en nuestro caso el DSP) pone la señal en "1" (Nivel Alto) en el bus de datos cuando está desocupado. *El inicio de la transmisión de datos (condición de inicio)* ocurrirá cuando el bus de datos esté en "0" (Nivel Bajo) y el bus de reloj este en "1". El manejo de las señales tanto del bus de reloj como del bus de datos, se realizan escribiendo al registro de control I2C descrito en la tabla 4.2 [24].

El dispositivo maestro siempre inicia las operaciones de lectura o escritura, y lo hace únicamente después de que ocurre la condición de inicio. La operación de escritura está completa solo cuando el maestro genera la condición de fin de transmisión, o bien, otra condición de inicio. *Para terminar una transmisión de datos, (condición de fin de transmisión)* el maestro retiene la señal del bus de datos en "1", mientras mantiene la señal de reloj en "1". La figura 4.8 muestra la gráfica de transmisión de datos por el bus SCCB [24].

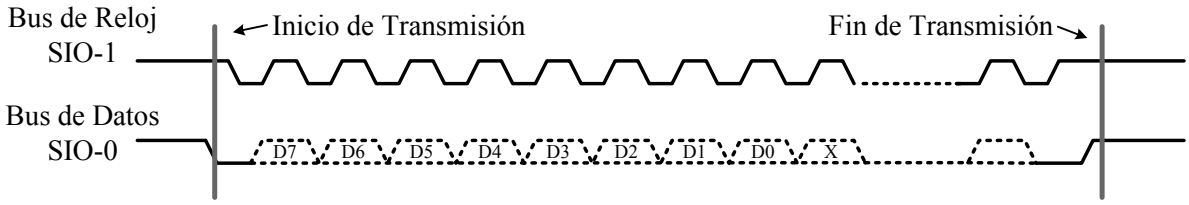


Figura 4.8 Diagrama de transmisión de datos por el bus SCCB.

Ciclos de transmisión

El elemento básico de la transmisión de datos es una ***fase***, como se ilustra en la figura 4.9. Una fase contiene un total de nueve bits. El 9º bit es un bit de *no importa* o un bit *NA*, dependiendo si se trata de un proceso de lectura o de escritura. El máximo número de fases que se pueden incluir en una transmisión es tres. El *Bit más significativo (MSB)* siempre es el primer bit al inicio de cada fase. Existen tres tipos de fases, las cuales se describen enseguida [24].

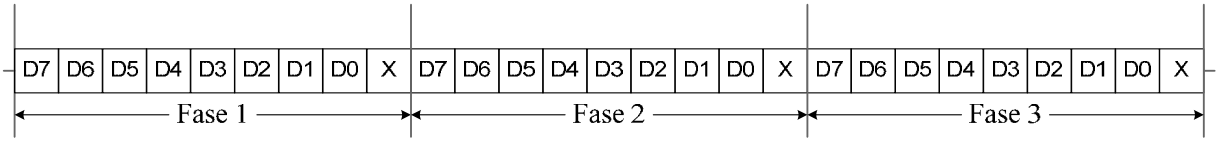


Figura 4.9 Esquema de las fases de transmisión de datos por el bus SCCB.

Fase uno: La fase uno solo puede ser insertada por el dispositivo maestro, y sirve para identificar el dispositivo esclavo al que se quiere acceder (puede haber varios esclavos). Cada dispositivo esclavo tiene una dirección única, esta dirección está compuesta por siete bits, ordenados del bit siete al bit uno, y puede identificar cerca de 128 dispositivos esclavos. El 8º bit, el bit cero, es el bit selector de lectura o escritura, el cual especifica la dirección de transmisión del ciclo actual. Un "0" lógico representa una operación de escritura y un "1" lógico representa una operación de lectura. El 9º bit de la fase uno debe ser un bit *no importa*.

Fase dos: La fase dos de transmisión puede ser insertada tanto por el dispositivo maestro como por el esclavo. Cuando una fase dos es insertada por el maestro, esta fase identifica la subdirección del dispositivo esclavo (registro) a la cual desea acceder el maestro. Cuando la fase dos es insertada por el esclavo, ésta contiene el dato a leer por el dispositivo maestro. El esclavo reconoce la subdirección del dato que se va a leer a través de una fase dos o fase tres del ciclo de transmisión de escritura, y el 9º bit es definido como un bit *NA.*

Fase tres: La fase tres solo puede ser insertada por el maestro. Esta fase contiene el dato que se va a escribir en el dispositivo esclavo. El 9º bit está definido como un bit *no importa*. El propósito del bit no importa es indicar que la transmisión está completa [24].

Por otro lado, existen tres tipos de ciclos de transmisión, que son:

- **Ciclo de transmisión de Escritura de tres fases**

El ciclo de transmisión de escritura de tres fases, es un ciclo de escritura completo en el cual el dispositivo maestro escribe un byte de datos a un dispositivo esclavo específico. La dirección ID identifica el dispositivo esclavo al que se desea acceder. La subdirección identifica la localización del registro al que se quiere escribir. El dato a escribir contiene ocho bits, y el dispositivo maestro sobrescribe el contenido de esta dirección específica. El 9º bit de las tres fases es el bit *no importa*. Este ciclo se ilustra en la figura 4.10 [24].

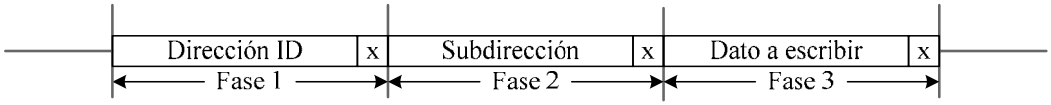


Figura 4.10 Ciclo de transmisión de escritura de tres fases.

- **Ciclo de transmisión de Escritura de dos fases**

La figura 4.11 muestra el ciclo de transmisión de escritura de dos fases, el cual es seguido por un ciclo de transmisión de lectura de dos fases. El propósito de utilizar un ciclo de escritura de dos fases es identificar la subdirección del dispositivo esclavo que se quiere leer en el siguiente ciclo de transmisión de lectura de dos fases. El 9º bit de las dos fases es el bit *no importa* [24].

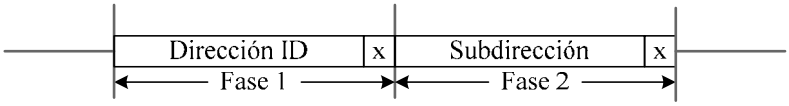


Figura 4.11 Ciclo de transmisión de escritura de dos fases.

- **Ciclo de transmisión de Lectura de dos fases**

Antes de un ciclo de lectura debe haber un ciclo de transmisión de escritura. El ciclo de lectura de dos fases no puede identificar la subdirección del esclavo, por lo que solo contiene la dirección ID del dispositivo esclavo y el dato a leer, como se ve en la figura 4.12 [24].

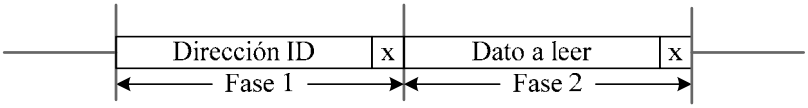


Figura 4.12 Ciclo de transmisión de escritura de dos fases.

Configuración de los registros de control del sensor video

Como se describió anteriormente, el sensor de video OV7620 cuenta con 124 registros de configuración. En la mayoría de los registros se mantienen los valores por default que proporciona el fabricante, por lo tanto solo describimos los registros más importantes configurados con valores diferentes a los que tienen por defecto.

Registro 11 – Control de reloj

Bits	SYN7	SYN6	CLK5	CLK4	CLK3	CLK2	CLK1	CLK0
Valor	0	1	0	0	0	1	0	0

SYN7 = 0, SYN6 = 1: HSYNC negativa y VSYNC negativa. En este campo del registro se específica que el pulso de sincronía vertical VSYNC se activa en el flanco de bajada.

Registro 12 – Control común A

Bits	COMA7	COMA6	COMA5	COMA4	COMA3	COMA2	COMA1	COMA0
Valor	1	0	0	0	0	0	0	0

COMA7 – Un “1” en este campo resetea el chip por software. Todos los registros, incluyendo el bus SCCB son puestos a sus valores por default. Después del reset, este bit se pone a “0”.

Registro 14 – Control común C

Bits	COMC7	COMC6	COMC5	COMC4	COMC3	COMC2	COMC1	COMC0
Valor	0	-	1	0	0	1	-	-

COMC5 – “1” Selecciona formato de salida digital QVGA – 240 x 320.

Registro 28 – Control común H

Bits	COMH7	COMH6	COMH5	COMH4	COMH3	COMH2	COMH1	COMH0
Valor	0	0	0	0	0	0	0	0

COMH5 – Selecciona modo de escaneo entrelazado

4.5 Diseño del software para la adquisición y transmisión de video al DSP

En la figura 4.13 se muestra el diagrama de flujo del sistema de adquisición y transmisión de video al DSP que diseñamos e implementamos en este trabajo. Los diferentes bloques del sistema se describen en las siguientes secciones de este capítulo y el bloque de compresión JPEG se describe en el *capítulo 5*.

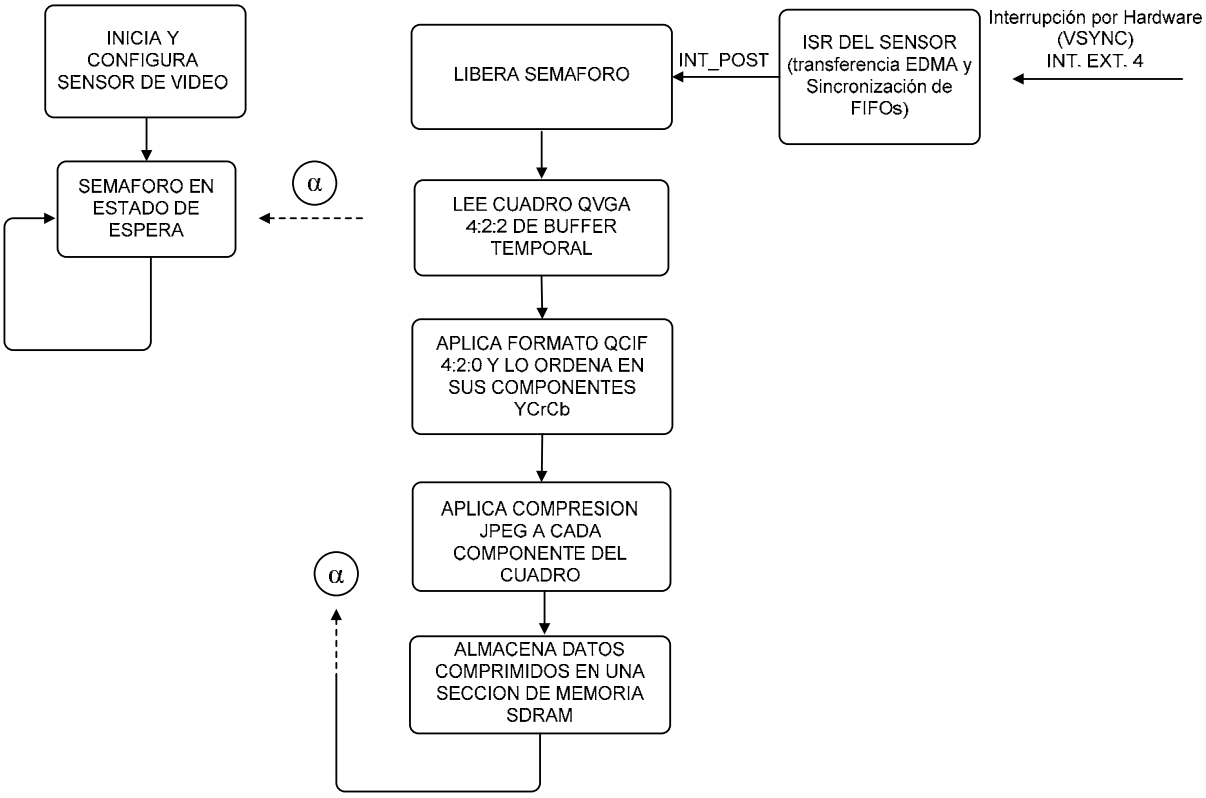


Figura 4.13 Diagrama de flujo del sistema de adquisición de video.

4.5.1 Rutina de comunicación I2C para configuración del sensor de video

Como se mencionó en la *sección 4.4*, lo primero que hace el sistema es configurar e iniciar los registros de control del sensor de video, a través del bus serial SCCB. En la figura 4.14 se muestra el diagrama de flujo de la rutina que realiza la comunicación I2C.

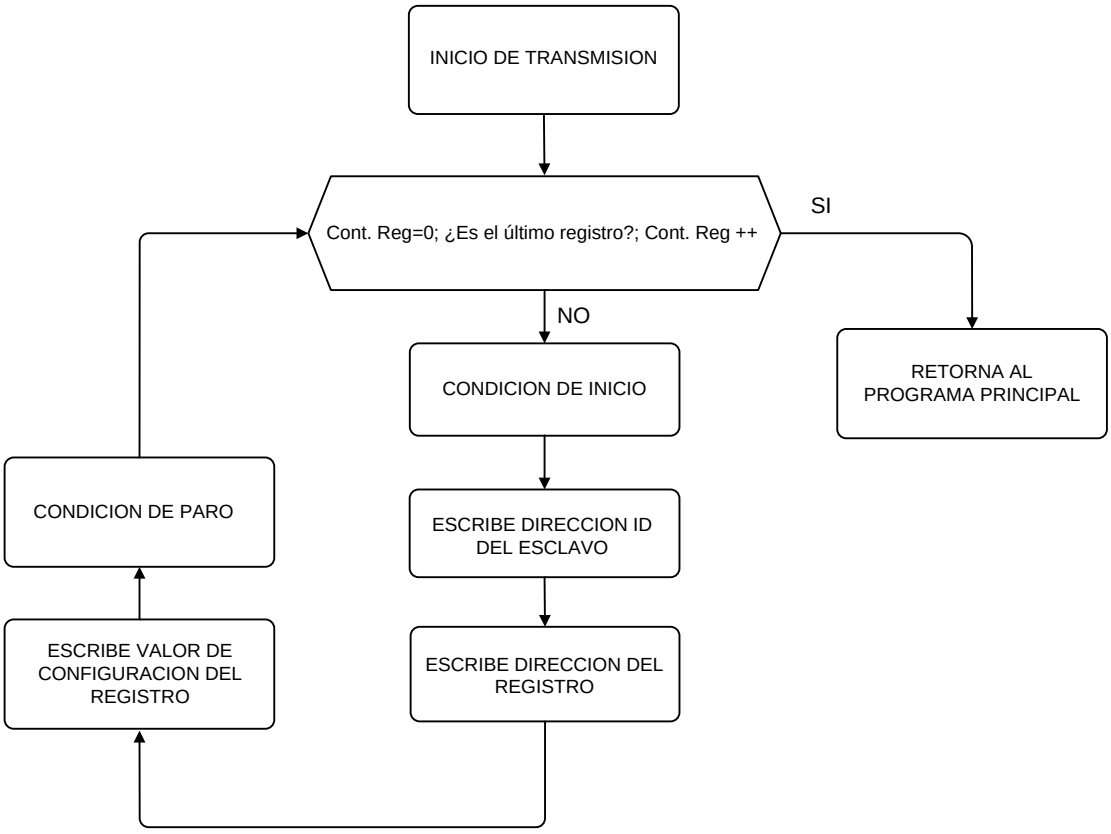


Figura 4.14 Diagrama de flujo para la comunicación I2C.

Al momento de iniciar la transmisión, la rutina para la comunicación I2C verifica si es el último registro a configurar, si no, genera una condición de inicio poniendo en "1" el bit *D1(I2CLK)* y en "0" el bit *D0(I2CDATA)* del registro de control I2C mapeado en la dirección de memoria 0xA0020000 (*sección 4.4.2*).

Una vez que se generó la condición de inicio, se procede a realizar un ciclo de transmisión de escritura de tres fases, escribiendo primero el byte con la dirección ID del esclavo, enseguida el byte con la dirección del registro y al final el byte con el valor de configuración. El proceso para realizar la escritura se describe más adelante.

Finalmente, una vez terminado el ciclo de transmisión de escritura de tres fases, se genera una condición de paro manteniendo en "1" tanto al bit *D1(I2CLK)* y como al bit *D0(I2CDATA)* del registro de control I2C. De esta manera, si no es el último registro, el ciclo comienza de nuevo, y si es el último registro, la rutina retorna al programa principal y el sensor queda configurado y listo para la captura de video.

El diagrama de flujo para realizar el proceso de escritura al bus I2C, se muestra en la figura 4.15.

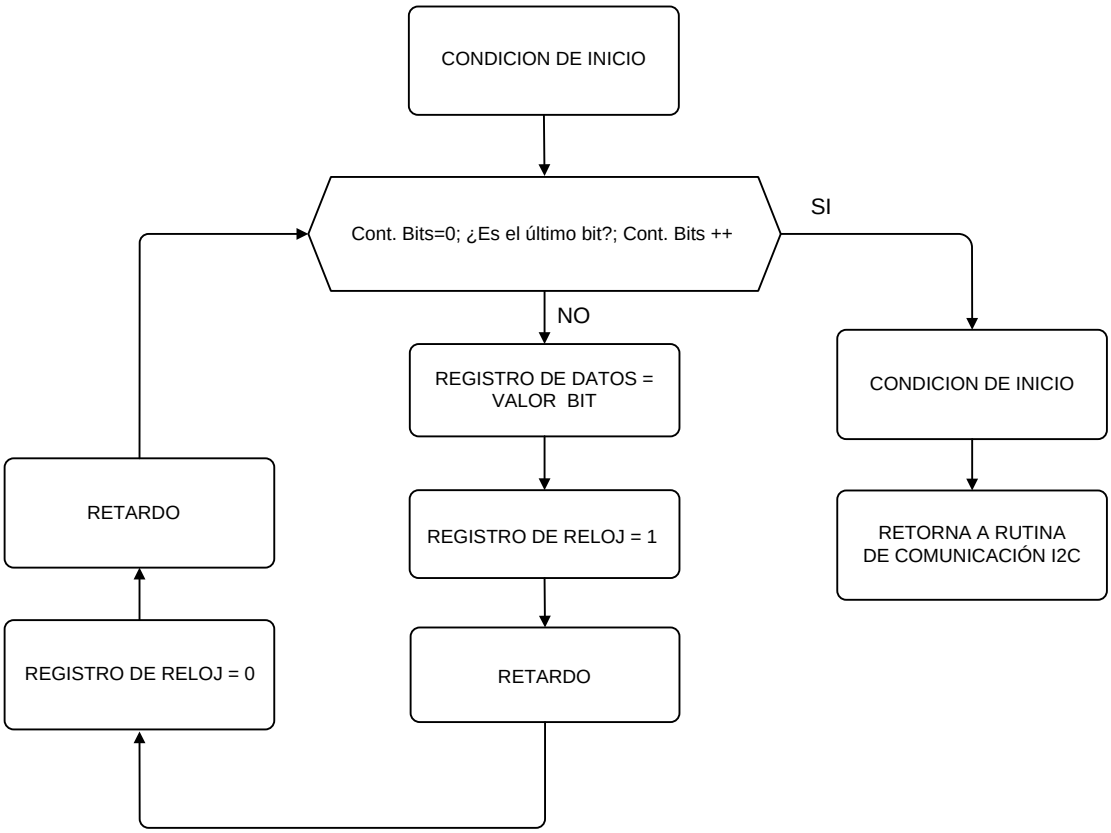


Figura 4.15 Diagrama de flujo para escribir al bus I2C.

Antes de empezar el proceso de escritura, se establece una condición de inicio. Posteriormente, la rutina de escritura verifica si es el último bit del byte que se va a transmitir, y si no es el caso, escribe el valor del bit correspondiente al campo *D0(I2CDATA)* del registro de control I2C. A continuación, la rutina pone en "1" el campo *D1(I2CLK)* del registro I2C, genera un retardo, pone en "0" nuevamente el campo *D1(I2CLK)* y genera otro retardo para que el bit de datos pueda ser escrito, como se mostró en la figura 4.8. Si el bit que se escribió no es el último del byte a escribir, el ciclo comienza de nuevo. De lo contrario, la rutina genera una condición de inicio y retorna a la rutina de comunicación I2C.

4.5.2 Rutina de Servicio de Interrupción del sensor

Como se observó en la figura 4.13, una vez que el sensor está inicializado, empieza la captura de imágenes. Mientras esto sucede, se utiliza un semáforo para dejar al programa principal en estado de espera hasta que sea liberado por la función *SEM_ipost*.

Una vez que el sensor adquiere un cuadro de video completo, genera un pulso de sincronía vertical *VSYNC* y causa una interrupción por hardware utilizando la *Interrupción Externa 4*. Cuando el DSP recibe la interrupción, ejecuta la *Rutina de Servicio de Interrupción del Sensor*, la cual realiza funciones de control del FIFO y dispara la transferencia EDMA.

Para controlar el buffer FIFO, el DSP utiliza el *registro de control FIFO* (el cual está mapeado en memoria en la dirección 0xA0028000) del CPLD de la DSKcam, y se describe en la tabla 4.3.

Posición del Bit	R/W	Función
D0 (FIFOHLD)	R/W	Esta señal previene a la cámara, para que el FIFO no se actualice sin ser necesario.
D1 (FIFOEMT)	R	Esta señal indica que el FIFO esta vacío.
D2 (FIFOSNC)	-	Este bit toma un valor '1' para causar una interrupción en cada pulso de Sincronía Vertical del sensor. Es utilizado para sincronizar el FIFO con la dirección de memoria 0.
D3 (Sin Uso)	-	

Tabla 4.3 Registro de Control FIFO.

El diagrama de flujo de la ISR del sensor se muestra en la figura 4.16 y su funcionamiento se explica enseguida.

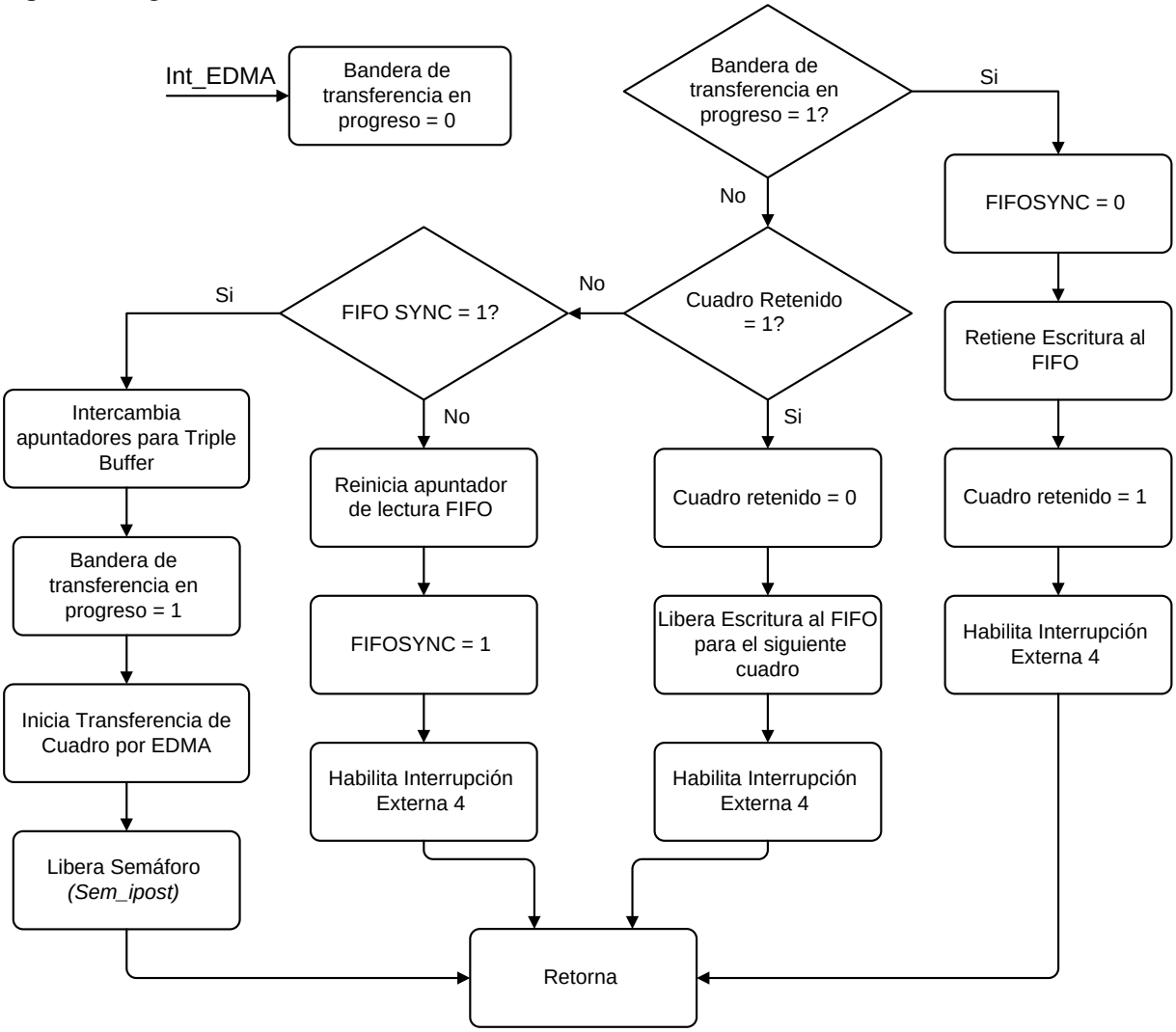


Figura 4.16 Diagrama a bloques de la *Rutina de Servicio de Interrupción del Sensor*.

En la figura 4.16 se observa que la rutina ISR primero verifica si existe una transferencia EDMA en progreso, comprobando la bandera de transferencia en progreso. Si la bandera está habilitada, se inhabilita la escritura al FIFO para evitar un desbordamiento de datos, se habilitan la bandera de cuadro retenido y la interrupción externa 4, y retorna. Para retener la escritura al FIFO se escribe un "1" al campo *D0(FIFOHLD)* del registro de control FIFO, mostrado en la tabla 4.3.

Por otro lado, si la bandera de transferencia en progreso es igual a cero, la rutina identifica si hay un cuadro retenido en el buffer FIFO, y si esto se cumple, se limpia la bandera de cuadro retenido, se libera al buffer FIFO para escribir el siguiente cuadro, se habilita la interrupción externa 4 y se retorna al programa principal. La escritura al FIFO se libera escribiendo un "0" al campo *D0(FIFOHLD)* del registro de control FIFO.

En el otro caso, si la bandera de cuadro retenido es 0, la rutina verifica el valor del campo *D2(FIFOSNC)* del registro de control FIFO. Si el valor de este campo es 0, se reinicia el apuntador de lectura del FIFO, se habilita la interrupción externa 4 y se retorna. En caso de que el valor del campo *D2(FIFOSNC)* sea 1, se intercambian apuntadores para el uso de un triple Buffer (el cual se explica más adelante), se habilita la bandera de transferencia en progreso y el DSP inicia la transferencia de datos por EDMA desde los buffers FIFO hasta una sección de memoria SDRAM. Posteriormente, la ISR del sensor libera al semáforo del programa principal que está en estado de espera mediante la función *sem_ipost*. Finalmente, en la figura 4.16 se observa que la bandera de transferencia en progreso toma el valor 0, cuando ocurre una interrupción de transferencia completa del canal EDMA.

Configuración de la transferencia EDMA

Para mover los datos de la imagen desde los buffers de memoria FIFO de la DSKcam a la memoria SDRAM de la tarjeta DSK6416, se utiliza un canal EDMA. Antes de explicar la configuración del canal EDMA es necesario conocer la forma en que se ordenan los datos de un cuadro de video después de que son transferidos desde el puerto de video digital del sensor a la memoria de los buffers FIFO, lo cual se muestra en la tabla 4.4. Los valores n y m son: n = 76,800 y m = n/2 = 38,400, donde n es el número de muestras de luminancia y m al número de muestras de crominancia (debido al submuestreo 4:2:2), y el tamaño de cada muestra es de 8 bits.

Imagen Completa
Y0
Cr0
Y1
Cb0
Y2
Cr1
Y3
Cb1
...
Yn-1
Crm
Yn
Cbm

← 8 bits →

Tabla 4.4 Orden de los datos en la memoria del buffer FIFO de una imagen completa.

La figura 4.17 ilustra la transferencia por EDMA desde los buffers FIFO a la memoria externa SDRAM del DSP y la tabla 4.5 muestra la configuración de los registros del canal EDMA.

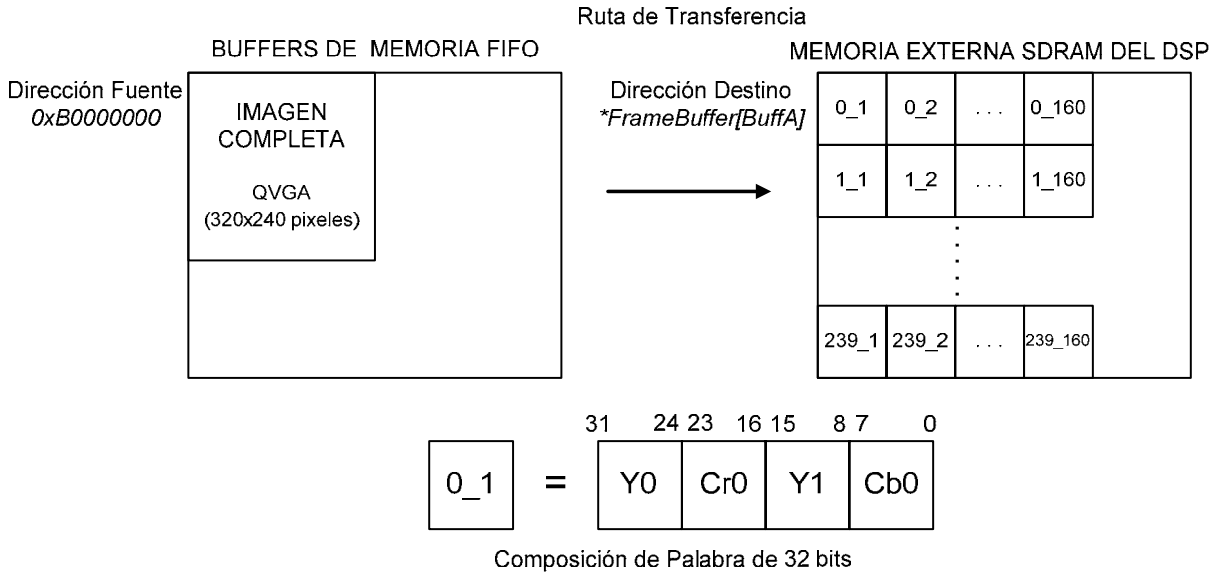


Figura 4.17 Esquema de la configuración de la transferencia por EDMA.

Parámetro EDMA_OPT_RMK	
Campo	**Definición**
EDMA_OPT_PRI_LOW	Prioridad de transferencia baja
EDMA_OPT_ESIZE_32BIT	Palabra de 32 bits
EDMA_OPT_2DS_YES	Fuente de datos bidimensional
EDMA_OPT_SUM_NONE	Dirección de la fuente de datos fija (no se modifica)
EDMA_OPT_2DD_YES	Destino de datos bidimensional
EDMA_OPT_DUM_INC	Incremento de la dirección de destino
EDMA_OPT_TCINT_YES	Indica cuando una transferencia está completa
EDMA_OPT_TCC_OF(10),	Bits más significativos código transferencia completa
EDMA_OPT_TCCM_OF(0),	Bits menos significativos código transferencia completa
EDMA_OPT_ATCINT_NO	Deshabilita señal de transferencia completa alternativa
EDMA_OPT_ATCC_OF(0),	Código de transferencia alterna
EDMA_OPT_PDTS_DISABLE	Deshabilita fuente de dispositivos periféricos
EDMA_OPT_PDTD_DISABLE	Deshabilita destino de dispositivos periféricos
EDMA_OPT_LINK_NO	No hay enlace de parámetros de eventos
EDMA_OPT_FS_YES	El canal está sincronizado por frame
Parámetro EDMA_SRC_OF(FIFO)	Dirección fuente FIFO = 0xB0000000
Parámetro EDMA_CNT_RMK(239,160)	239 indica el número de vectores - 1 en un bloque 2D, 160 indica el número de elementos en un vector
Parámetro EDMA_DST_OF(FrameBuffer[BuffA])	*FrameBuffer[BuffA]* es un apuntador que indica la dirección de memoria del destino de los datos
Parámetro EDMA_IDX_RMK(0x0000,0x0000)	Indice de Frame e índice de elementos
Parámetro EDMA_RLD_RMK(0x0000,0x0000)	Usado para recarga de parámetros

Tabla 4.5 Estructura para la configuración del canal EDMA.

De acuerdo con la tabla 4.5, al estar deshabilitado el enlace de parámetros de eventos, y las direcciones de destino y fuente de dispositivos periféricos, el CPU es el que inicia la transferencia EDMA. El tipo de transferencia es bidimensional (2D). En la figura 4.17 y la tabla 4.5 se observa que la fuente de datos es el FIFO, el cual está mapeado en la dirección 0xB0000000. Como no se incrementa la dirección de destino, se transmite cada elemento del bloque de la fuente de datos a la dirección destino. Como se observa en la figura 4.17 cada palabra es de 32 bits, y está compuesta por una muestra de luminancia Y0, una de crominancia roja Cr0, otra de luminancia Y1 y finalmente una muestra de crominancia azul Cb0. Como se utiliza el submuestreo 4:2:2, cada palabra está compuesta por 2 pixeles (*sección 1.5.2*), y dado que el tamaño del cuadro QVGA es de 320 pixeles x 240 líneas, solo necesitamos la mitad del número de elementos para transferir un cuadro completo.

4.5.3 Uso de Triple Buffer

Para evitar que los datos de las imágenes se traslapen en la memoria SDRAM y mejorar el desempeño del sistema, se implementó un *triple Buffer*. Esta técnica de triple Buffer se realiza mediante la actualización de apuntadores, y cada buffer puede almacenar un cuadro de video completo. Para la transferencia y ordenamiento de las imágenes tenemos tres casos:

Al inicio, los tres apuntadores de los buffers A, B, y C están en el ***caso 1***. El apuntador A señala al Buffer 1, el B al Buffer 2 y el C al Buffer 3. Cuando se llama a la *Rutina de Servicio de Interrupción del Sensor* ésta intercambia a los apuntadores A y B, después de eso el apuntador A señala al Buffer 2 y el apuntador B al Buffer 1. Al momento de hacer la transferencia de datos de la imagen desde los buffers FIFO a una sección de memoria SDRAM mediante un canal EDMA, la *ISR del Sensor* siempre toma como dirección destino la que señala el apuntador A. En este caso los datos del cuadro de video se escribirán en el Buffer 2, como se muestra en la figura 4.18a. Por otro lado, mientras se realiza la transferencia de datos al Buffer 2, la aplicación principal intercambia los apuntadores B y C, de tal modo que el apuntador B señala al Buffer 3 y el apuntador C señala al Buffer 1. El DSP siempre procesa la imagen señalada por el apuntador C, que en este caso es el Buffer 1.

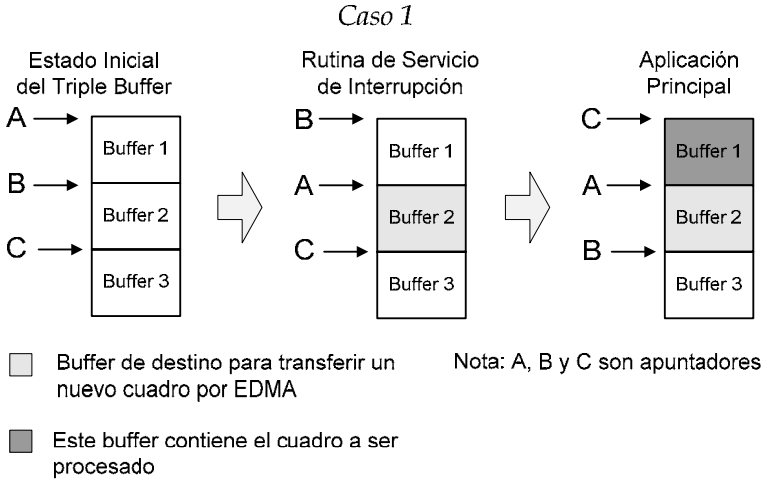


Figura 4.18a Esquema del triple Buffer, caso 1.

En el ***caso 2***, el cual se muestra en la figura 4.18b, el apuntador A señala al Buffer 2, B señala al Buffer 3 y C señala al Buffer 1. Cuando se ejecuta la *Rutina de Servicio de Interrupción del Sensor*, esta intercambia los apuntadores A y B, de modo que el apuntador A señala al Buffer 3 y B señala al Buffer 2. En este caso, el nuevo cuadro de video será escrito en el Buffer 3, el cual es señalado por el apuntador A. Mientras se transfieren los datos al Buffer 3, la aplicación principal intercambia los apuntadores B y C. Ahora B señala al Buffer 1 y C señala al Buffer 2. Por lo tanto, el DSP procesará el cuadro señalado por el apuntador C, el cual está en el buffer 2.

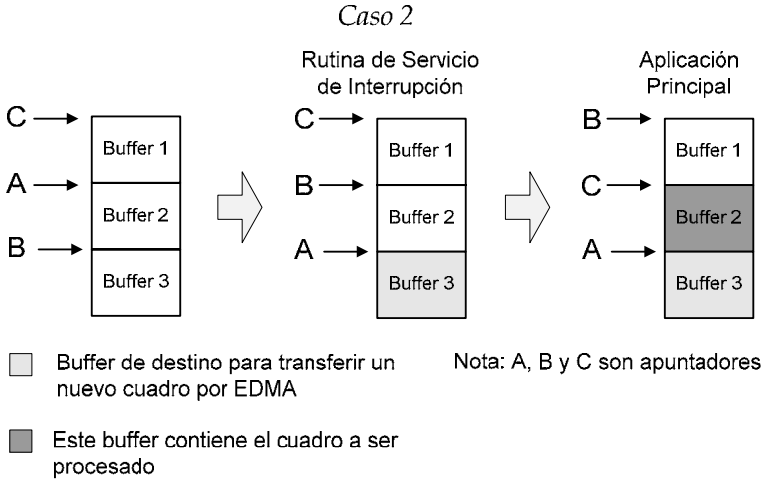


Figura 4.18b Esquema del triple Buffer, caso 2.

En el ***caso 3***, el apuntador A señala al Buffer 3, B señala al Buffer 1 y C señala al Buffer 2, como se ilustra en la figura 4.18c. Cuando se ejecuta la *Rutina de Servicio de Interrupción del Sensor*, esta intercambia los apuntadores A y B, de modo que ahora el apuntador A señala al Buffer 1 y B señala al Buffer 3. En este caso, el nuevo cuadro de video será escrito en el Buffer 1, el cual es señalado por el apuntador A. Mientras se transfieren los datos al Buffer 1, la aplicación principal intercambia los apuntadores B y C. Ahora B señala al Buffer 2 y C señala al Buffer 3. El DSP procesará el cuadro del buffer 3, el cual está señalado por el apuntador C. En este caso, podemos notar que los apuntadores vuelven al estado inicial como en el caso 1, para volver a iniciar el ciclo.

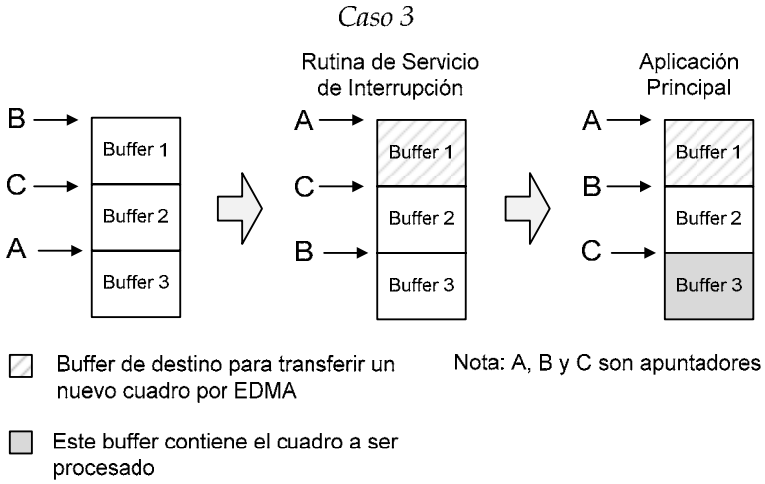


Figura 4.18c Esquema del triple Buffer, caso 3.

4.6 Adaptación del formato de las imágenes de video a QCIF 4:2:0

Como se mostró en la figura 4.13, una vez que se libera el semáforo, el programa principal lee un cuadro en formato QVGA 4:2:2 del buffer temporal de la memoria SDRAM para aplicarle el formato QCIF 4:2:0 y ordenar cada componente por separado, como se muestra en la figura 4.19. El propósito de utilizar el formato QCIF es que el número de unidades mínimas de codificación a procesar (*sección 3.4*) se reduzca de 1200 MCUs para QVGA a 396 MCUs para QCIF. Por otro lado, al utilizar submuestreo 4:2:0 la información de la crominancia se reduce en un 50%.

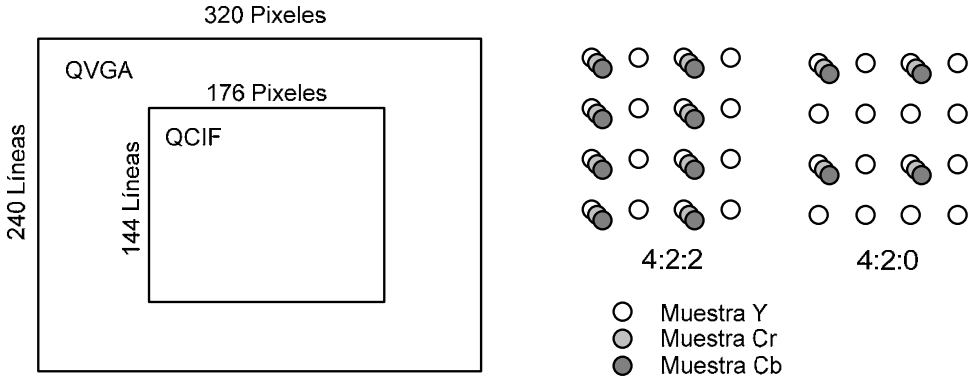


Figura 4.19 Cambio de formato de imagen QVGA 4:2:2 a una imagen QCIF 4:2:0.

En la tabla 4.6 se muestra como están ordenadas las muestras de forma entrelazada en el buffer temporal, y como quedan las muestras después de aplicarles el submuestreo 4:2:0

Cuadro sin ordenar
Y0
Cr0
Y1
Cb0
Y2
Cr1
Y3
Cb1
Y4
Cr2
Y5
Cb2

Cuadro ordenado
Y0
Y1
Y2
Y3
...
Cr0
Cr2
...
Cb0
Cb2
...

Tabla 4.6 Cuadros de video sin ordenar y ordenado.

Una vez que el cuadro de video está en el nuevo formato, se aplica el proceso de compresión JPEG a cada componente de la imagen y los datos comprimidos de la imagen se almacenan en una sección de la memoria SDRAM. Posteriormente, el programa regresa al bloque de semáforo en espera, hasta que otro cuadro esté listo para ser procesado. En el siguiente capítulo se describe la implementación de la compresión y descompresión MJPEG.

RESUMEN

En este capítulo, se hizo una descripción general del sistema de adquisición de video MJPEG. Además, se describió el hardware utilizado en el proyecto, comenzando con las características del procesador digital de señales DSP C6416, incluyendo canales EDMA para la transferencia de datos, el procesamiento de interrupciones, la tarjeta de desarrollo DSK6416, la cual se utiliza para programar al DSPC6416 y la sincronización con semáforos en el DSP. También se hizo el análisis de la tarjeta de expansión DSKcam y del sensor de video CMOS conectado a esta tarjeta.

Por otra parte, se mencionó la implementación del sistema de adquisición de video tanto a nivel hardware como a nivel software. En la implementación del software, se describió la implementación de la comunicación I2C para programar los registros del sensor, el uso de la Rutina de Servicio de Interrupción del Sensor para iniciar la transferencia de imágenes desde el sensor de video a la memoria externa del DSP, la implementación de un triple buffer para optimizar el funcionamiento del sistema y la adaptación del formato de las imágenes de video a un formato QCIF 4:2:0, para disminuir la cantidad de información a procesar. En el capítulo 5 se describe como se implementó el proceso de compresión MJPEG en el DSP.

5.- IMPLEMENTACION DEL COMPRESOR DE VIDEO MJPEG EN EL DSP C6416

El objetivo de este capítulo es presentar la implementación del método de compresión de video MJPEG, tomando como base los fundamentos teóricos expuestos en los capítulos 1 al 3, además de que es la segunda parte del sistema de adquisición y compresión de video implementado en este trabajo, el cual se mostró en el capítulo 4. Este capítulo se divide en cuatro secciones. En la primera sección se presenta como se implementó el sistema de compresión de video MJPEG. La segunda sección presenta las etapas que integran el proceso de descompresión de los datos comprimidos para reconstruir el video original y poder visualizarlo en una PC. La tercera sección describe como se puede implementar la transmisión de los datos comprimidos desde la memoria externa del DSP a una PC a través de un puerto serial. La última sección presenta como se puede realizar la sincronización de las tareas del sistema, desde la captura de video hasta la visualización del video reconstruido.

5.1 Implementación del algoritmo de compresión de video MJPEG en el DSP

Como se mencionó en el *capítulo 3*, MJPEG es un método de compresión no estandarizado que aplica la compresión JPEG a cada cuadro de una secuencia de video. En este trabajo se decidió implementar el método de compresión MJPEG ya que es muy utilizado en sistemas de edición de vídeo, videocámaras IP y en video conferencias de baja calidad en tiempo real. Además, se eligió MJPEG ya que presenta las siguientes ventajas (*sección 3.1*):

- Con MJPEG, al procesar las imágenes de manera independiente, se limita el efecto de error, ya que si se presenta un error en una imagen, éste no se propaga en las siguientes imágenes como en el caso de MPEG-1 y MPEG-2.

- MJPEG proporciona tasas de compresión entre 10:1 y 15:1 sin afectar significativamente la calidad visual de las imágenes.

- MJPEG proporciona buenos resultados en tiempo real con equipos de bajo costo.

- La implementación de MJPEG es relativamente simple, la complejidad del algoritmo, los requisitos de hardware, procesamiento y almacenamiento son muy bajos en comparación con los codecs *Interframe*.

- El estándar JPEG es un estándar muy popular y es de dominio público.

Implementación del modo básico JPEG en la compresión MJPEG

El *modo básico* JPEG es un subconjunto del modo secuencial JPEG con codificación entrópica Huffman y está diseñado para adaptarse a un amplio número de aplicaciones (*sección 3.3, 3.4*). *Las imágenes codificadas en modo básico solo pueden tener muestras de 8 bits, y sus tablas de cuantización y Huffman son más pequeñas que en las imágenes en modo secuencial extendido.*

Para el desarrollo de este trabajo se eligió el modo básico secuencial JPEG con codificación Huffman, debido a su simplicidad en comparación con los otros modos JPEG, además, un decodificador que puede manejar el modo secuencial extendido es capaz de manejar el modo básico de forma transparente.

Como se mostró en la figura 3.5 *(secc. 3.4)* para realizar la compresión MJPEG, a cada imagen de la secuencia de video se le aplica el algoritmo de compresión JPEG. Además, en el caso de las imágenes a color, la compresión JPEG se aplica a cada componente de color de la imagen.

En el capítulo 4 se mostró como se adquieren las imágenes de una secuencia de video. Cabe recordar que las imágenes se almacenan en una sección de memoria externa SDRAM del DSP, en un formato QCIF con submuestreo 4:2:0, y sus componentes YCrCb se ordenan por separado, de tal manera que las imágenes quedan listas para la compresión JPEG, como se muestra en la figura 5.1.

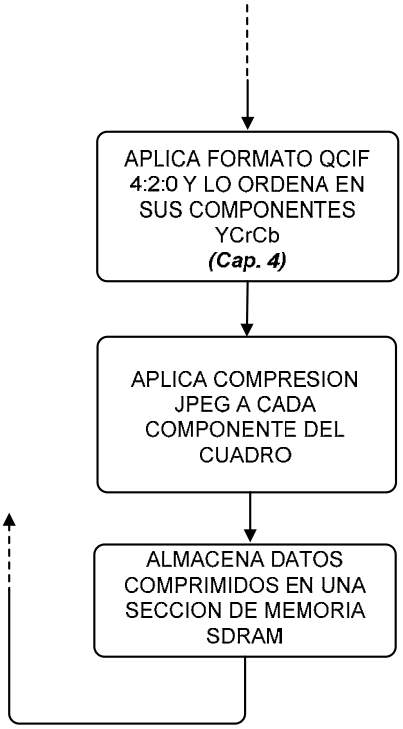


Figura 5.1 Sección del diagrama de flujo del sistema de adquisición y compresión de video.

En la figura 5.2 se muestra el diagrama de flujo para realizar la compresión JPEG de cada una de las imágenes de video, teniendo como resultado la compresión MJPEG. En las siguientes secciones se describe como se implementaron las etapas de compresión JPEG de la figura 5.2.

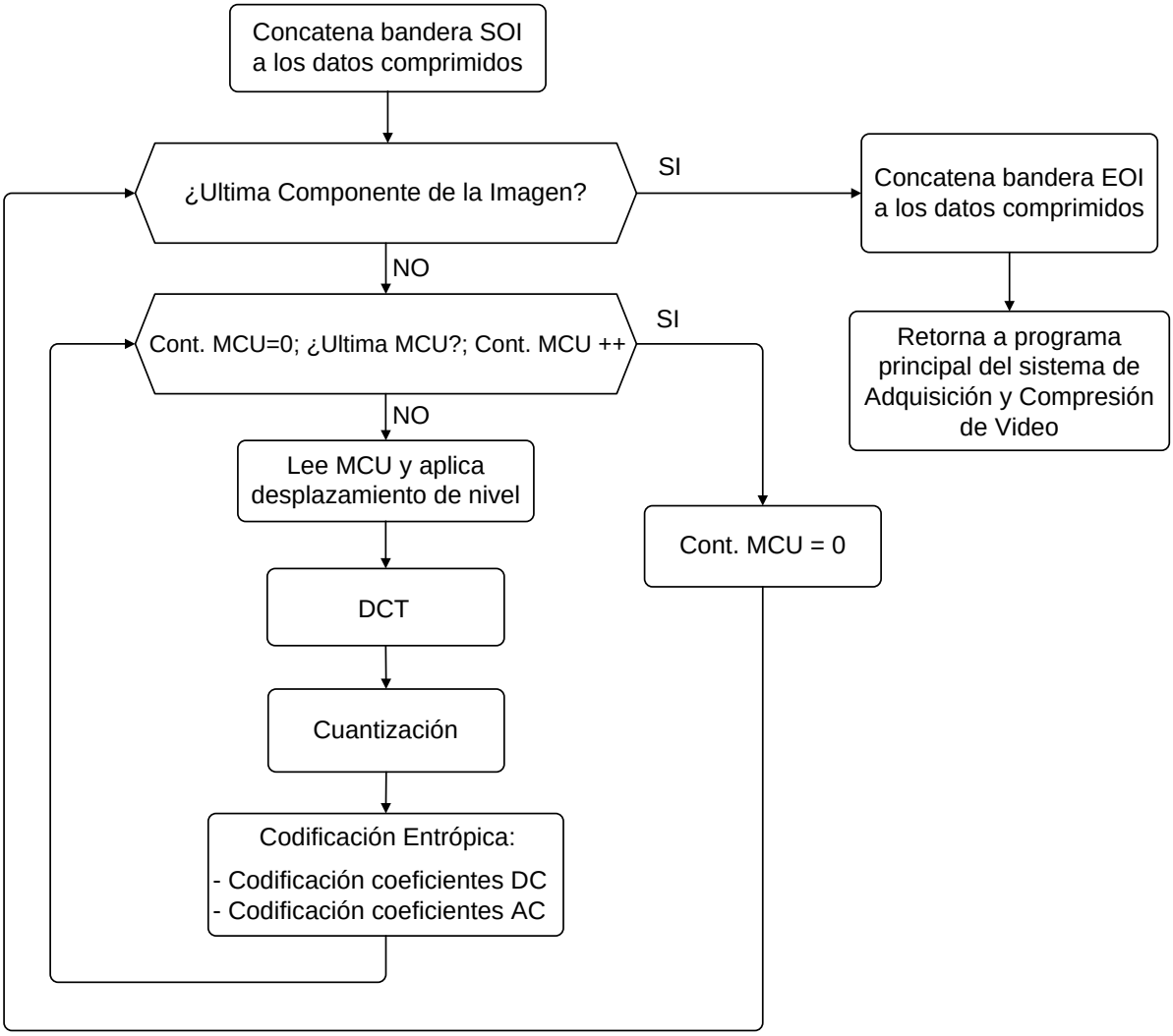


Figura 5.2 Diagrama de flujo de compresión JPEG.

5.1.1 Preparación de las unidades mínimas de codificación MCU

Como se explicó en la *sección 3.4*, cada componente de color de una imagen se procesa en bloques de 8x8 muestras, a los que llamamos MCUs. El orden de codificación de bloques es de izquierda a derecha y de arriba abajo. La figura 5.3 muestra el orden de codificación de las unidades de datos MCU para el componente de luminancia de una imagen en tamaño QCIF.

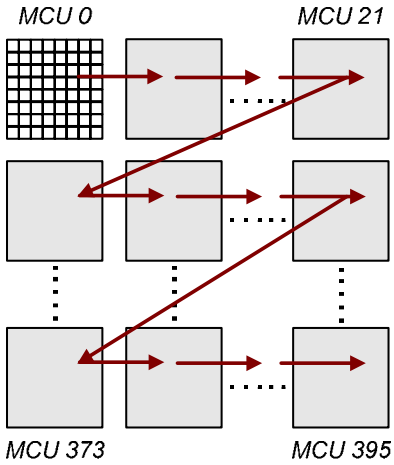


Figura 5.3 Orden de codificación de las unidades MCU de la luminancia de una imagen QCIF.

Como se describió en la *sección 4.6*, cuando capturamos una imagen y le aplicamos el formato QCIF 4:2:0, los datos de cada componente de la imagen se ordenan de forma separada en la memoria externa SDRAM del DSP, así que por simplicidad en el programa se decidió que las unidades de datos no estuvieran entrelazadas *(sección 3.4)*.

Cuando se obtienen las MCUs para la luminancia, el número de MCUs es de 396 MCUs, en cambio para los componentes de crominancia Cr y Cb, el número total de MCUs es de 99 MCUs por cada imagen debido al uso del submuestreo 4:2:0.

Con la finalidad de analizar cómo se implementó la compresión JPEG, tomaremos como referencia la primera unidad MCU del componente de luminancia de una imagen obtenida por la cámara y que ha sido adaptada al formato QCIF con submuestreo 4:2:0, y mostraremos como se realizó cada etapa del proceso de compresión. La tabla 5.1 muestra los valores de la primera unidad MCU de una imagen en su forma original, lista para que se le apliquen los procesos de compresión de JPEG que se describen en las siguientes secciones.

161	161	161	164	161	161	161	162
162	161	162	164	161	161	162	162
162	161	163	164	162	162	162	163
163	163	164	164	162	162	162	163
164	164	165	165	163	163	163	165
164	164	165	165	163	163	164	164
165	164	165	167	164	165	163	166
165	165	166	167	164	165	164	166

Tabla 5.1 Unidad MCU original.

5.1.2 Desplazamiento de nivel

Como se explicó en la *sección 3.4.1*, el primer paso en el proceso de compresión JPEG es el desplazamiento de nivel y consiste en restar el valor de 128 a cada elemento de la unidad de datos como se ilustra en la tabla 5.2.

33	33	33	36	33	33	33	34
34	33	34	36	33	33	34	34
34	33	35	36	34	34	34	35
35	35	36	36	34	34	34	35
36	36	37	37	35	35	35	37
36	36	37	37	35	35	36	36
37	36	37	39	36	37	35	38
37	37	38	39	36	37	36	38

Tabla 5.2 Unidad MCU después del desplazamiento de nivel

5.1.3 Aplicación del algoritmo optimizado de la DCT a las unidades MCU

Una vez que se obtiene la unidad de datos desplazada en nivel, se procede a aplicar la DCT. Como se mencionó en la *sección 2.4.1*, la DCT de dos dimensiones se puede calcular a partir de dos transformadas DCT unidimensionales. Para esto, es necesario aplicar la DCT unidimensional a las filas del bloque de 8x8 para obtener una matriz intermedia. Posteriormente, sobre la matriz obtenida se aplica nuevamente la transformada unidimensional, pero ahora sobre las columnas para obtener la matriz de coeficientes de la DCT bidimensional. El algoritmo para el cálculo de la DCT unidimensional implementado es el propuesto por Wang, Suehiro y Hatori, el cual fue seleccionado para el proyecto ya que es un algoritmo óptimo que reduce el número de operaciones considerablemente [20].

La figura 5.4 muestra el algoritmo optimizado basado en Wang, Suehiro y Hatori *(sección 2.4.2)*. En la figura, los datos de entrada son las variables *x(0)* a *x(7)* y los coeficientes de la DCT son las variables *X(0)* a *X(7)*, además las flechas indican la dirección en que fluye la información, en caso de que las flechas estén modificadas por un factor la información se multiplica por este factor, y por último, cuando dos cantidades (flechas) se encuentran en un punto estas se suman.

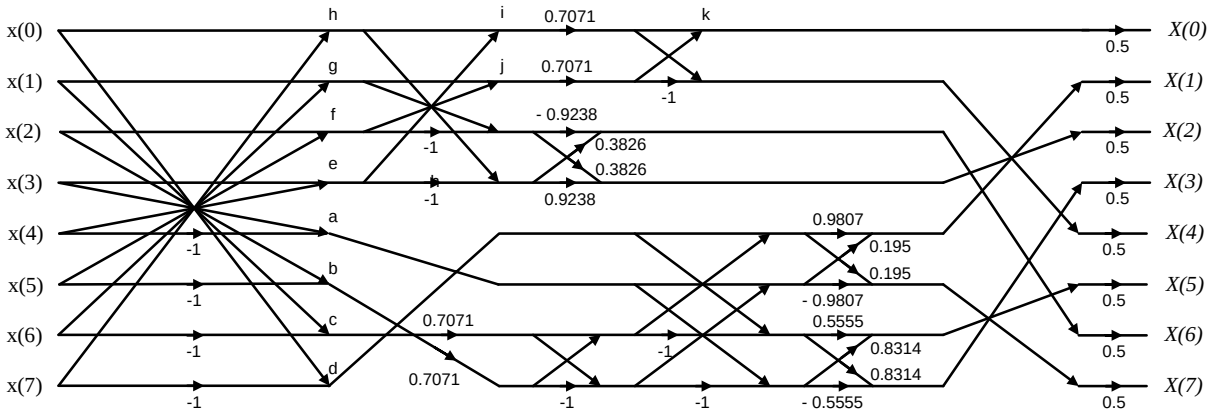


Figura 5.4 Algoritmo para el cálculo de la DCT propuesto por Wang, Suehiro y Hatori.

Para mostrar un ejemplo de cómo se calculan los coeficientes tomemos como ejemplo el cálculo del coeficiente de la DCT *X(0)*. En la figura 5.4 podemos observar que para poder llegar al coeficiente DCT *X(0)*, los datos de entrada *x(0)* y *x(7)* se unen en el punto *h*, por lo que el valor de *h* queda definido por la ecuación (5.1):

$$h = x(0) + x(7) \tag{5.1}$$

Posteriormente, siguiendo el flujo de las flechas para llegar a *X(0)* en la figura 5.3 el valor *h* se suma con el valor *e* en el punto *i*, y tenemos las ecuaciones (5.2) y (5.3):

$$i = h + e \tag{5.2}$$
$$e = x(3) + x(4) \tag{5.3}$$

el valor *i* se multiplica por el factor *0.7071* y se encuentra en el punto *k* con el factor *j*, por lo que se tienen las ecuaciones 5.4 a la 5.7:

$$k = i*0.7071 + j*0.7071 = 0.7071(i + j) \tag{5.4}$$
$$j = g + f \tag{5.5}$$
$$g = x(1) + x(6) \tag{5.6}$$
$$f = x(2) + x(5) \tag{5.7}$$

Finalmente, en la figura 5.3 se observa que para obtener *X(0)* el valor de *k* se multiplica por 0.5. Sustituyendo las ecuaciones (5.5) y (5.2) en la ecuación (5.4) y multiplicando *k* por 0.5 el valor de *X(0)* se define por la ecuación (5.8):

$$X(0) = 0.3536*(e + f + g + h) \tag{5.8}$$

A continuación el conjunto de ecuaciones (5.9) y (5.10) describe el algoritmo completo basado en Wang, y Suehiro y Hatori.

$$\begin{aligned} h &= x(0) + x(7) \\ g &= x(1) + x(6) \\ f &= x(2) + x(5) \\ e &= x(3) + x(4) \\ d &= x(0) - x(7) \\ c &= x(1) - x(6) \\ b &= x(2) - x(5) \\ a &= x(3) - x(4) \end{aligned} \tag{5.9}$$

$$X(0) = 0.3536*(e + f + g + h) \tag{5.10}$$
$$X(1) = 0.4904*(d + (0.7071*c) + (0.7071*b)) + 0.0975*(a + (0.7071*c) - (0.7071*b))$$
$$X(2) = 0.4619*(h - e) + 0.1913*(g - f)$$
$$X(3) = 0.2778*((0.7071*c) - (0.7071*b) - a) + 0.4157*(d - (0.7071*c) - (0.7071*b))$$
$$X(4) = 0.3536*(e + h - f - g)$$
$$X(5) = 0.2778*(d - (0.7071*c) - (0.7071*b)) + 0.4157*(a - (0.7071*c) + (0.7071*b))$$
$$X(6) = 0.1913*(h - e) + 0.4619*(f - g)$$
$$X(7) = 0.0975*((0.7071*c) + (0.7071*b) + d) + 0.4904*((0.7071*b) - (0.7071*c) - a)$$

Para obtener los coeficientes de la DCT bidimensional de la unidad de datos MCU desplazada en nivel (tabla 5.2), como primer paso se toman los elementos de la primera fila como muestras de entrada al algoritmo optimizado, con lo que se obtienen los coeficientes DCT unidimensionales. Estos coeficientes corresponden a la primera fila de la matriz intermedia. Posteriormente, se procede a aplicar la DCT unidimensional a las siguientes filas de la matriz original, con lo cual se obtiene la matriz intermedia procesada por filas, como se muestra en la tabla 5.3.

94.7648	-0.1979	-0.9238	-1.2491	1.4144	0.9693	-0.3826	-1.5687
95.8256	0.1546	-0.4619	-1.2263	1.0608	1.835	-0.1913	-0.7777
97.24	-0.4333	-0.8445	-1.3642	1.0608	1.1415	0.7325	-0.3848
99.3616	0.791	-0.8445	-0.8657	0.3536	0.0228	0.7325	-0.3525
101.8368	0.6759	0.2706	-2.0496	0.7072	0.2582	0.6532	-0.5247
101.8368	0.7506	0	-1.5363	0	1.0264	0	-0.1494
104.312	0.2178	-0.5739	-1.3466	1.768	0.4789	1.3857	-1.8465
105.3728	0.4956	-0.3826	-1.837	0.7072	0.5764	0.9238	-1.4308

Tabla 5.3 Matriz intermedia de la DCT bidimensional procesada por filas.

Una vez que se obtienen todas las filas de la matriz intermedia, se procede a aplicar la DCT optimizada, pero ahora tomando como muestras de entrada los elementos de la primera columna de la matriz. Después de que se aplica la DCT, los datos de salida corresponden a la primera columna de la matriz final de coeficientes DCT. Este proceso continúa hasta que se procesan todas las columnas de la matriz intermedia. El resultado es la matriz final de coeficientes DCT bidimensional, la cual se muestra en la tabla 5.4.

283.0746	0.8679	-1.3297	-4.0575	2.5007	2.2307	1.3627	-2.4876
-10.2484	-0.6841	-0.5622	0.5016	0.3131	0.7654	-1.085	0.3281
-0.2871	-0.5295	-0.3749	-0.0162	0.8282	0.6121	-0.3017	-1.3801
-0.6405	0.2665	0.488	-0.1807	-0.059	0.0401	-0.7705	-0.094
0.7502	0.3801	0	-0.1867	-0.2501	-0.9388	0	-0.254
-0.2626	-0.2293	-0.7512	0.6132	0.4997	-0.6425	0.5148	-0.5138
-0.6929	-0.2491	-0.0518	-0.184	-0.6137	0.1745	-0.3749	0.5594
0.6263	-0.5987	0.1119	-0.4851	0.8798	-0.1752	0.5763	-0.4927

Tabla 5.4 Matriz final de coeficientes de la DCT bidimensional.

5.1.4 Proceso de Cuantización

Como se describió en las *secciones 2.5.1 y 3.4.3,* una vez que se le aplica la DCT a la MCU y se tienen los 64 coeficientes de la transformada (tabla 5.4), éstos deben ser cuantizados, de acuerdo a un tamaño de escalón, el cual está definido por tablas de cuantización. Estas tablas están basadas en estudios sobre imágenes, y son diferentes para la luminancia y las crominancias, y se pueden consultar en las tablas *A.5* y *A.6* del *anexo A.*

Para modificar el tamaño de escalón al momento de la implementación, se necesitarían de varias tablas de cuantización, lo que requeriría más memoria para su almacenamiento. Sin embargo, en vez de utilizar varias tablas de cuantización, se decidió utilizar un factor de cuantización el cual definimos como ***QF***. Este factor *QF* permite escalar los valores de cada coeficiente en las tablas de cuantización, modificando el tamaño de escalón. Como se mencionó en las *secciones 2.3.1 y 3.4.3,* un tamaño de escalón muy grande genera una compresión más alta, pero con el riesgo de tener una mayor distorsión al decodificar la imagen. El estándar JPEG recomienda que los valores del factor *QF* estén entre 0.2 y 3 [19].

Si modificamos la ecuación (3.1) de la *sección 3.4.3* para variar el valor del tamaño de escalón utilizando el factor *QF* y una sola tabla de cuantización, obtenemos la ecuación (5.11):

$$Sq_{uv} = round\left(\frac{S_{uv}}{Q_{uv}/QF}\right) \qquad (5.11)$$

donde: $\boldsymbol{S_{uv}}$ es el *valor del coeficiente original.*
$\boldsymbol{Sq_{uv}}$ es el *valor del coeficiente cuantizado*
$\boldsymbol{Q_{uv}}$ es el *valor de la tabla de cuantización*
$\boldsymbol{QF}$ es el *factor de cuantización*

Con base en los resultados obtenidos en otros trabajos, los valores de *QF* utilizados en este proyecto son 2, 0.5 y 0.2, ya que con estos resultados se obtienen calidades de imagen muy buena, regular y pobre respectivamente [16], [19].

Por otro lado, la ecuación (5.11) implica el uso de divisiones, las cuales son muy pesadas para procesar. Con el fin de optimizar los procesos, se decidió utilizar multiplicaciones en lugar de las divisiones. Esto se realiza modificando las tablas de cuantización (*tablas A.5* y *A.6 del anexo A*), sustituyendo cada elemento por su inverso (tablas 5.5 y 5.6) y modificando la ecuación (5.11) para obtener la ecuación (5.12), la cual nos dará el mismo resultado que la ecuación (5.12) sin utilizar divisiones.

$$Sq_{uv} = round\left(S_{uv} \bullet Q_{uv}^{-1} \bullet QF\right) \qquad (5.12)$$

donde: $\boldsymbol{Q_{uv}^{-1}}$ es el *tamaño de escalón de cuantización*.

Las tablas de cuantización inversas para la luminancia y crominancias utilizadas en este proyecto se muestran en las tablas 5.5 y 5.6 respectivamente.

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 5.5 Tabla de cuantización inversa 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 5.6 Tabla de cuantización inversa para las crominancias.

Para cuantizar la matriz de coeficientes DCT de la tabla 5.4, le aplicamos la ecuación 5.12, donde cada coeficiente DCT se multiplica por el valor en la misma posición de la tabla de cuantización inversa de luminancia, ya que la MCU original pertenece a dicho componente, y por un factor de cuantización QF=2. La matriz de coeficientes DCT cuantizados se muestran en la tabla 5.7.

35	0	0	-1	0	0	0	0
-2	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Tabla 5.7 Matriz de coeficientes DCT cuantizados

En la tabla 5.7 se observa que para un bloque de 8x8 coeficientes, solo quedan 3 coeficientes diferentes de cero, es decir el 4.69 % de los coeficientes del bloque, los cuales son los más significativos.

5.1.5 Codificación Huffman de los datos cuantizados

En la *sección 3.4.4* se analizó la codificación entrópica de los datos cuantizados utilizando la codificación Huffman, la cual codifica los coeficientes DC y AC de forma diferente.

Ordenamiento zigzag

El primer paso para realizar la codificación Huffman es ordenar los datos cuantizados en un vector unidimensional siguiendo el arreglo zigzag *(sección 3.4.4)*. El primer coeficiente del ordenamiento en zigzag es el coeficiente DC, y los restantes son los coeficientes AC. El algoritmo del ordenamiento zigzag se divide en dos etapas, las cuales se muestran a continuación:

En la primera etapa se realiza el ordenamiento mostrado en la figura 5.5. Para realizar este arreglo, se utiliza un apuntador para leer los valores de la matriz de datos cuantizados, siguiendo el flujo de la figura 5.5. El valor inicial del apuntador es el primer coeficiente del bloque de 8x8 *(coeficiente DC)*. El apuntador se debe 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, +8, +1, +7, +7, +7, +7, +7, +7, +7

donde los signos + indican que el apuntador se debe incrementar *n* elementos de izquierda a derecha y de arriba a abajo, y el signo – indica un decremento de *n* elementos de derecha a izquierda y de abajo hacia arriba. En la figura 5.6 se muestra el diagrama de flujo del algoritmo para implementar la primera etapa del ordenamiento zigzag.

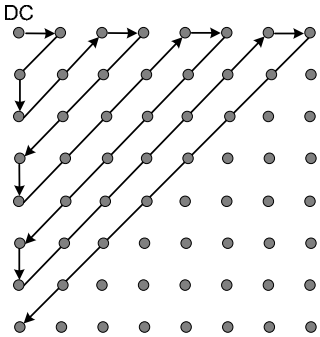


Figura 5.5 Primera etapa del ordenamiento zigzag.

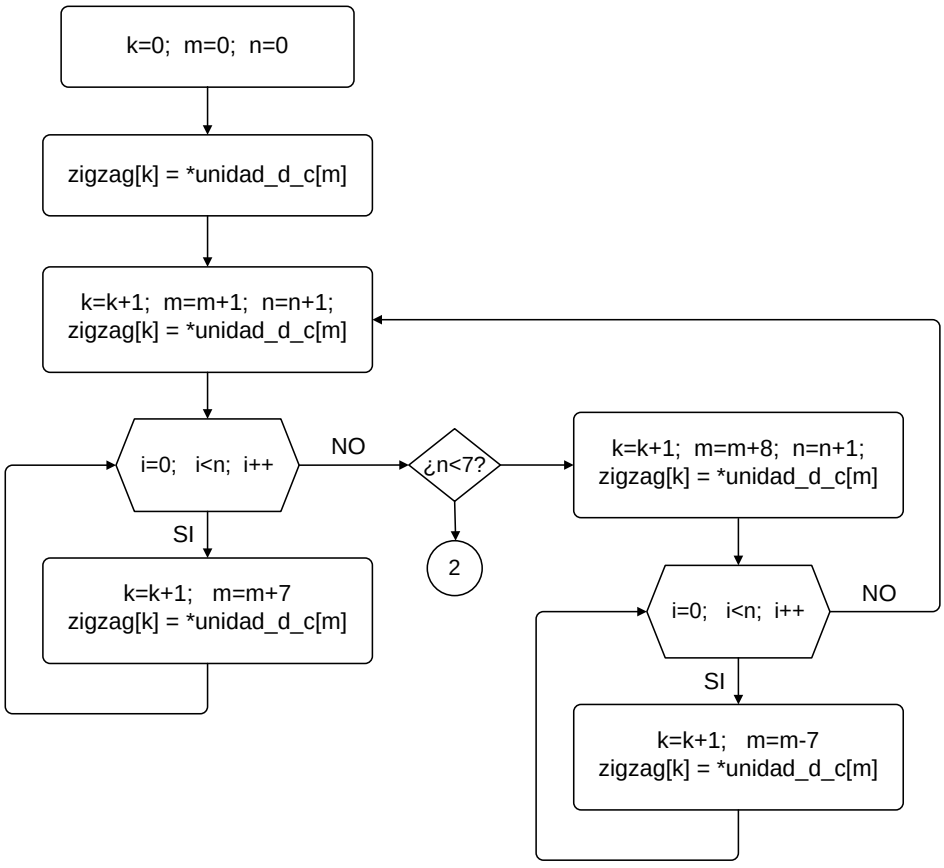


Figura 5.6 Algoritmo de la primera etapa de ordenamiento zigzag.

En la figura 5.6 la variable *k* especifica el elemento del arreglo *zigzag[ ]*, la variable *m* el elemento de la unidad de datos cuantizados *unidad_d_c[ ]*, y la variable *n* es una variable de control del algoritmo.

En la segunda etapa se realiza el ordenamiento mostrado en la figura 5.7. Para realizar este arreglo, el apuntador se modifica siguiendo el flujo de la figura 5.7. El apuntador se incrementa o decrementa de acuerdo a la siguiente 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.

La figura 5.8 muestra el diagrama de flujo para la segunda etapa del arreglo zigzag.

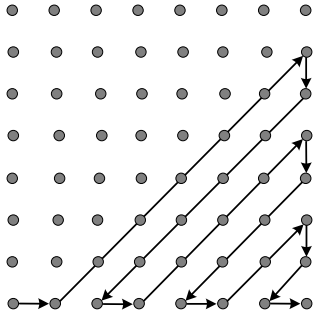


Figura 5.7 Segunda etapa del ordenamiento zigzag.

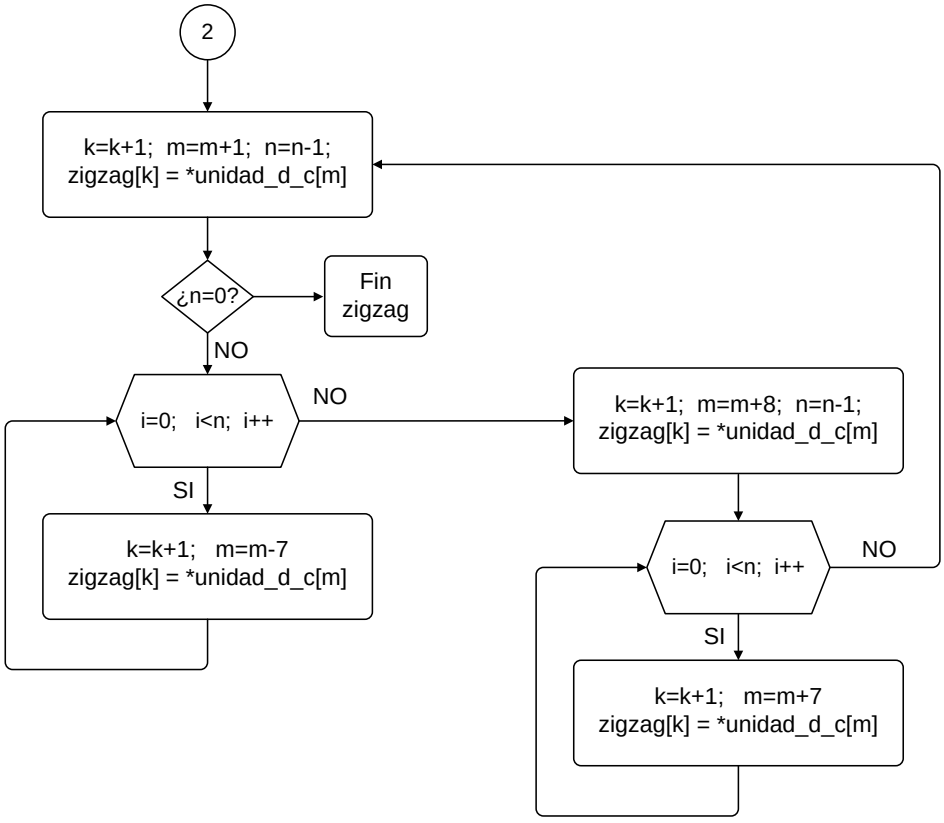


Figura 5.8 Algoritmo de la segunda etapa de ordenamiento zigzag.

En la figura 5.8 la variable *k* especifica el elemento del arreglo *zigzag[ ]*, la variable *m* especifica el elemento de la unidad de datos cuantizados *unidad_d_c[ ]* y la variable *n* es una variable de control del algoritmo.

Por otro lado, si aplicamos el algoritmo zigzag a los coeficientes cuantizados de la tabla 5.7, obtenemos el arreglo zigzag de la ecuación 5.13, donde el primer valor corresponde al coeficiente DC y los restantes a los coeficientes AC.

zigzag[64] = [35, 0, -2, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] (5.13)

Codificación de los coeficientes DC

En la *sección 3.4.4* se explicó que en la codificación DPCM de los coeficientes DC de la DCT, el valor que se codifica es la diferencia DPCM (*DIFF*) entre el coeficiente DC_i cuantizado de la unidad de datos actual y el coeficiente DC_{i-1} de la unidad de datos anterior, del mismo componente. En el arreglo zigzag de la ecuación 5.13, el valor del coeficiente DC es 35, y como la matriz original (tabla 5.1) es la primera unidad de datos del componente de luminancia, entonces el valor anterior de DC es 0, por lo tanto de acuerdo al ecuación 3.2 de la *sección 3.4*, el valor DPCM es:

$$DPCM = 35 - 0 = 35 \quad (5.14)$$

Esta diferencia DPCM pertenece a la categoría 6 de la tabla 5.8, por lo tanto el código Huffman que le corresponde es 1110. Además, los bits adicionales que se concatenan al código Huffman son los seis bits del valor 35 en binario, es decir 100011, por lo que el resultado final será el valor de *1110100011*. *(Las tablas completas se muestran en las tablas 3.4 y 3.5 de la sección 3.4.4).*

Categoría SSSS	**Diferencia DPCM (DIFF)**		**Código Huffman**	**SSSS bits adicionales**	
.....	.....		.....	.....	.....
5	-31, ..., -16,	16, ..., 31	110	00000, ...,01111,	10000, ..., 11111
6	-63, ..., -32,	32, ..., 63	1110	000000, ...,011111,	100000, ..., 111111
7	-127, ..., -64,	64, ..., 127	11110	0000000, ...,0111111,	1000000, ..., 1111111
.....	.....	.....	.....	.....	.....

Tabla 5.8 Clasificación de diferencias DPCM, códigos Huffman y bits adicionales.

Una vez codificada la diferencia DPCM se procede a codificar los coeficientes AC, lo que se muestra en la siguiente sección.

Codificación de los coeficientes AC

Como se mostró en la *sección 3.4.4*, el primer paso para la codificación de los coeficientes AC es el ordenamiento zigzag de los coeficientes cuantizados. Ya que muchos de los coeficientes AC se vuelven cero después de la cuantización, el codificador Huffman combina las *longitudes de ceros* (definidas con el valor *"RRRR"*) con las *categorías SSSS* de los coeficientes distintos de cero. Las longitudes de ceros indican el número de ceros que preceden a un coeficiente distinto de cero en la secuencia zigzag.

En el arreglo zigzag de la ecuación 5.13, el primer valor del coeficiente AC diferente de 0 es -2, el cual corresponde a la categoría SSSS = 2 de acuerdo a la tabla 5.9, y es precedido por un coeficiente de valor cero, entonces se tiene un valor de longitud de ceros RRRR = 1, por tal motivo le corresponde el código base 0x12 localizado en la fila 1 y la columna 2 de la tabla 5.10. Al código base 0x12 le corresponde el código Huffman *11011* de la tabla 5.11. *(Las tablas completas se muestran en la tabla 3.7 y 3.8 de la sección 3.4.4 y en el anexo A).*

Categoría SSSS	Coeficientes AC	
.....	.....	
1	-1,	1
2	-3, -2,	2, 3
3	-7, ..., -4,	4, ..., 7
.....	.....	.....

Tabla 5.9 Clasificación de los coeficientes AC distintos de cero en categorías SSSS.

		SSSS															
													*	*	*	*	*
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
RRRR	**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
	...	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
	15	**ZRL**	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF
***** No son utilizadas en el modo básico **N/A** No aplican para el modo secuencial																	

Tabla 5.10 Codificación Huffman de los coeficientes AC y sus longitudes de ceros.

Categoría	Longitud de Código	Código Huffman
*0x00 (**EOB**)*	*4*	*1010*
.....	.....	.....
0x12	*5*	*11011*
0x13	*8*	*11111001*
.....	.....	.....
0x31	*6*	*111010*
.....	.....	.....
*F0 (**ZRL**)*	*11*	*11111111001*

Tabla 5.11 Códigos base para la generación de coeficientes AC de la luminancia.

Una vez que se obtiene el valor del código Huffman, se procede a concatenar los bits adicionales. Como el valor del coeficiente diferente de cero es -2, el cual pertenece a la categoría SSSS = 2, y es un valor negativo, al código Huffman se le concatenan los 2 bits menos significativos del valor 2 en binario en complemento a 1, teniendo como resultado el valor 01. Por lo tanto, el primer coeficiente AC codificado es *1101101*.

El segundo coeficiente AC diferente de 0 es -1, al cual le corresponde la categoría SSSS=1 de la tabla 5.9 y es precedido por tres ceros, por tal motivo, le corresponde el valor 0x31 en la tabla 5.10. El código Huffman proporcionado por la tabla 5.11 es el valor *111010*. Como el valor del coeficiente AC es -1, con categoría SSSS = 1 y valor negativo, al código Huffman 111010 se le concatenan el bit menos significativos de 1 en binario en complemento a 1 (el valor 0) lo que genera que el segundo coeficiente AC codificado sea el valor *1110100*.

Finalmente, como todos los coeficientes restantes del arreglo zigzag tienen valor 0, tenemos un *fin de bloque (sección 3.4.4)*. Al fin de bloque le corresponde el valor 0x00 de la tabla 5.10, y le corresponde el código Huffman 1010 de la tabla 5.11. En la tabla 5.12 se muestran los valores AC codificados; esta tabla proporciona el valor del coeficiente AC, su categoría SSSS correspondiente, el número RRRR de ceros que le anteceden, su código Huffman, los bits adicionales y por último el valor del coeficiente codificado.

Coeficiente	SSSS	RRRR	RRRR/SSSS	Código Huffman	SSSS bits adicionales	Coeficientes Codificados
-2	2	1	0x12	11011	01	11011
-1	1	3	0x31	111010	0	1110100
EOB	-	-	-	1010	-	1010

Tabla 5.12 Coeficientes AC codificados.

La secuencia final codificada de la unidad de datos MCU, considerando el coeficiente DC y los coeficientes AC codificados, queda de la siguiente manera:
1110 1000 1111 0110 1111 0100 1010

Los datos codificados se almacenan en palabras de 16 bits, tomando en cuenta que en aplicaciones de almacenamiento de datos se recomienda que la codificación y la decodificación se realicen empleando *buffers de memoria* con una longitud que sea múltiplo de 8 bits. Con esto, cuando el buffer está lleno, se puede almacenar la secuencia de bits aprovechando una palabra de un byte o múltiplo de un byte *(sección 2.7.2)*. Además, se utiliza el formato *big-endian* que es el que maneja el estándar JPEG *(sección 3.2)*, por lo que el orden en que se almacena la secuencia de datos anterior (datos comprimidos) en la memoria SDRAM se muestra en la tabla 5.13.

Dirección	0x8008BC52	0x8008BC54	...
Dato	E8F6	F4A_	...

Tabla 5.13 Orden en memoria de los datos comprimidos utilizando formato big-endian.

El número total de bits requeridos para representar una unidad de datos MCU de 8x8 es solo 42, en comparación con los 512 que se necesitan originalmente. Por lo tanto se tiene un porcentaje de compresión de (42/512) x 100% = 8.2%, lo cual es una tasa de compresión muy buena.

El procedimiento de codificación de las unidades de datos MCU desde el desplazamiento de nivel hasta la codificación entrópica continúa para todas las MCU restantes, hasta completar todo un componente. Posteriormente se realiza la inserción de marcadores de fin de componente y de fin de imagen, lo que se describe en la siguiente sección.

5.1.6 Concatenación de marcadores JPEG

De acuerdo a la *sección 3.4.5,* los códigos marcadores se insertan dentro de la secuencia de datos codificados entropicamente. Los marcadores tienen una longitud de dos bytes, donde el primer byte siempre tiene el valor hexadecimal 0xFF. El segundo byte contiene el código que especifica el tipo de marcador. La única regla estricta del estándar JPEG es que un archivo debe iniciar con un marcador *Inicio de imagen (SOI)* y terminar con un marcador *Fin de imagen (EOI).* A un marcador SOI le siguen los datos comprimidos. Los marcadores JPEG utilizados en este trabajo se muestran en la tabla 5.14.

Valor	Símbolo utilizado en el estándar JPEG	Descripción
FFD8	SOI	Inicio de Imagen
FFD9	EOI	Fin de Imagen

Tabla 5.14 Marcadores JPEG utilizados en el proyecto.

Una vez que se ha terminado de comprimir una imagen, el algoritmo MJPEG regresa el control al programa principal, hasta que otra imagen está lista para ser comprimida como se mostró en la figura 4.14 de la *sección 4.5*.

5.2 Descompresión de video MJPEG en el DSP

Para recuperar el video comprimido y que pueda ser visualizado se necesita realizar la descompresión, el diagrama a bloques de la descompresión se mostró en la sección 3.4.6. El diagrama de flujo para la descompresión JPEG aparece en la figura 5.9. Este proceso se repite para reconstruir cada imagen de la secuencia de video a partir de los datos comprimidos. En las siguientes secciones se explican cada una de las etapas de la descompresión JPEG.

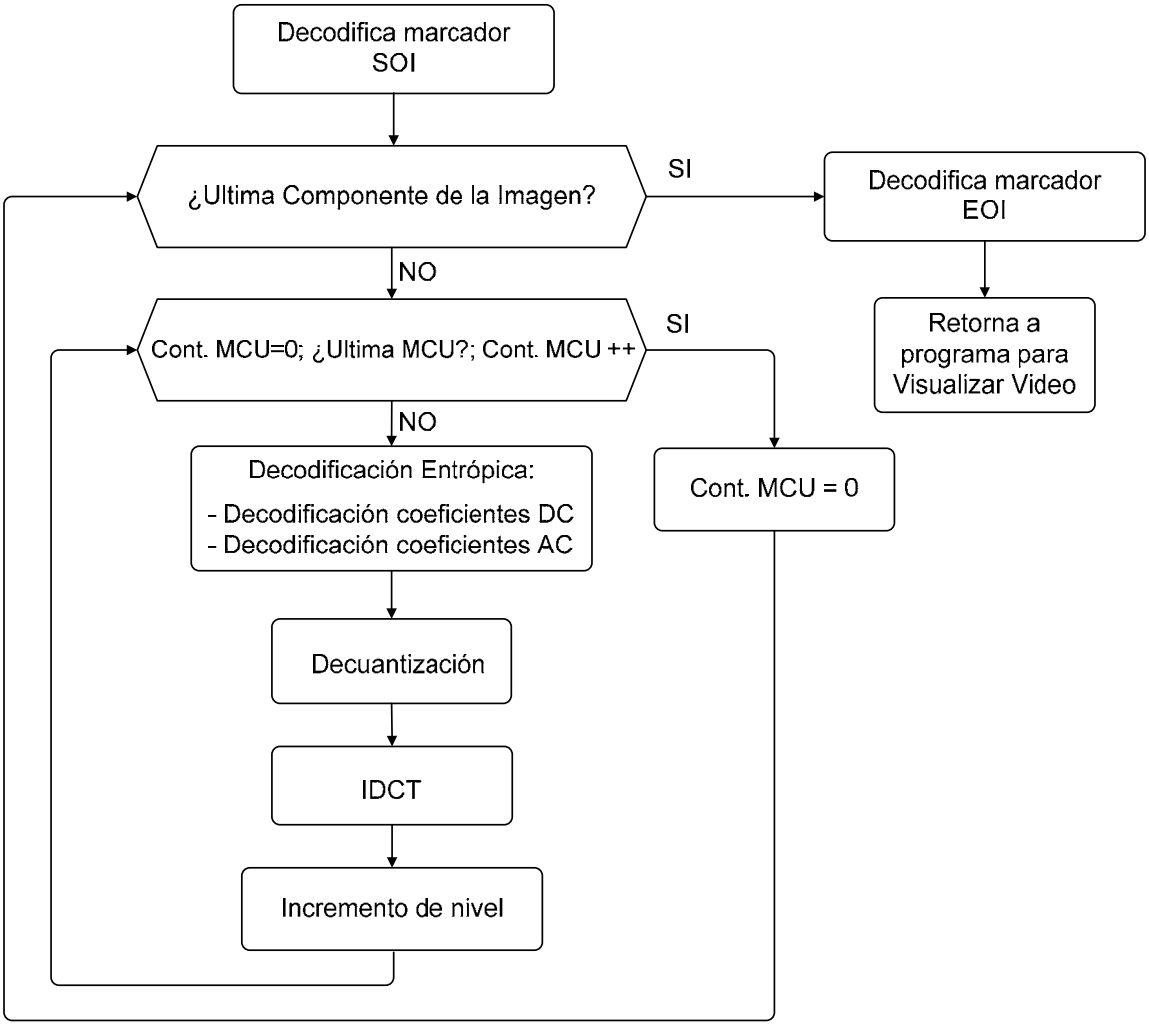


Figura 5.9 Diagrama de flujo de la descompresión JPEG.

Decodificación de marcador SOI

Antes de empezar la descompresión de los datos, el programa lee los primeros 16 bits y los compara con el valor de los marcadores. Si es un marcador SOI, pasa a la decodificación entrópica de los marcadores.

5.2.1 Decodificación Huffman de los datos comprimidos

Una de las características de los códigos de Huffman es que se puede decodificar la secuencia de datos a partir de la tabla de códigos, sin necesidad de señalizar el principio y el fin de un símbolo. Para ilustrar el procedimiento de la decodificación Huffman consideremos la cadena de bits *1110 1000 1111 0110 1111 0100 1010...* obtenida en la *sección 5.1.5* que corresponde a una MCU codificada. El primer paso para la decodificación Huffman es obtener el valor de la diferencia DPCM, lo cual se muestra en la siguiente sección.

Decodificación de los coeficientes DC

Al inicio de la decodificación, se comprueba si el primer bit *(1)* es una palabra del código Huffman, buscándolo en la tabla 5.15 de palabras del código Huffman con un solo bit. Si no coincide con ningún código, se comprueba si los dos primeros bits *(11)* pertenecen a algún código, y se continua con este procedimiento hasta que se encuentra la palabra *1110*, la cual se encuentra en la columna de códigos Huffman, y corresponde a la categoría SSSS=6.

Tabla de Códigos			
Categoría SSSS	**Diferencia DPCM (DIFF)**		**Código Huffman**
0	0		00
1	-1,	1	010
2	-3, -2,	2, 3	011
3	-7, ..., -4,	4, ..., 7	100
4	-15, ..., -8,	8, ..., 15	101
5	-31, ..., -16,	16, ..., 31	110
6	-63, ..., -32,	32, ..., 63	1110
7	-127, ..., -64,	64, ..., 127	11110
.....	.....	.....	.....

Secuencia de bits

1110*10001111011011110100101 0...*
1110 → C_6 = 1110

100011*1101101111101001010...*
100011→Bits adicionales de 1110

la siguiente cadena de bits se decodifica como coeficientes AC:
110110111101001010...

Tabla 5.15 Decodificación DPCM de una secuencia de bits.

Como a los valores del código Huffman al momento de la compresión se le concatenan los SSSS bits menos significativos del valor de la diferencia *(sección 3.4.4)*, en el caso de la categoría SSSS=6, el numero de bits concatenados al código Huffman es de 6. Por lo tanto, los siguientes 6 bits que le siguen al valor de código Huffman 1110 corresponden a los bits adicionales, y es la palabra *100011*. Como se mencionó en la sección 3.4.4 y se mostró en la tabla 3.5 de esta sección, el primero de los bits adicionales es 1 si la diferencia DPCM es positiva, y 0 si la diferencia es negativa. Por consiguiente, el valor de la diferencia DPCM es *35*, el cual es el valor decimal de *100011*.

Ahora, para encontrar el valor del coeficiente DC de la unidad de datos MCU aplicamos la ecuación 5.15, la cual se obtiene despejando DC_i de la ecuación 3.2 de la *sección 3.4.4*.

$$DC_i = DPCM + DC_{i-1} \tag{5.15}$$

Si consideramos que la unidad de datos MCU que se está reconstruyendo es la primera MCU de la imagen, entonces el valor del coeficiente DC_{i-1} es 0. Por lo que el coeficiente DC de esta MCU es 35, y este valor se almacena como el primer elemento en el arreglo unidimensional *zigzag_rec[ ]*.

Decodificación de los coeficientes AC

El siguiente paso en la decodificación Huffman es la decodificación de los coeficientes AC. Este proceso es muy parecido a la decodificación de coeficientes DC. Para ilustrar la decodificación de coeficientes AC tomaremos la siguiente cadena de bits de este ejemplo *110110111101001010...* la cual se obtuvo según la tabla 5.15. Al inicio se comprueba si el primer bit *(1)* es una palabra del código Huffman, buscándolo en la tabla 5.16 de palabras del código Huffman con un solo bit. Si no coincide con ningún código, se comprueba si los dos primeros bits *(11)* pertenecen a algún código, y se continua con este procedimiento hasta que se encuentra la palabra *11011* que se encuentra en la columna de códigos Huffman, y corresponde a la categoría 0x12.

Tabla de Códigos Huffman	
Categoría	**Código Huffman**
*0x00 (**EOB**)*	*1010*
.....	*.....*
0x12	*11011*
.....	*.....*
0x31	*111010*
.....	*.....*
*F0 (**ZRL**)*	*11111111001*

Secuencia de bits

110110111101001010...

11011 → C_{0x12} = 11011

0111101001010...

01 → Bits adicionales de 11011

11101001010...

Tabla 5.16 Decodificación de los coeficientes AC de una secuencia de bits.

Después de que se obtiene la categoría 0x12, se procede a encontrar el valor RRRR de la longitud de ceros y la categoría SSSS del coeficiente diferente de cero utilizando la tabla 5.10 de la sección 5.1.5. Los valores para la categoría 0x12 son *RRRR=1* y *SSSS=2*. Así como en el caso de la decodificación de coeficientes DC, el número de bits adicionales es *SSSS=2*, por lo que los bits adicionales del código Huffman 11011 son los dos bits que le siguen a este valor en la cadena de bits, es decir, los bits (01). Como el primer valor de los bits es un cero, entonces el valor del coeficiente AC es negativo, y si aplicamos el complemento a 1 a estos bits, obtenemos el valor del coeficiente AC, el cual es *-2*.

Antes de almacenar el valor del coeficiente AC, se tiene que el valor RRRR es igual 1, por lo que al coeficiente AC le precede un 0. Así el arreglo zigzag_rec queda de la siguiente forma:

$$zigzag\_rec[64] = [35, 0, -2, ...] \tag{5.16}$$

Para recuperar los coeficientes restantes se procede de forma similar. La cadena de bits que queda por decodificar es 11101001010. El siguiente código Huffman que se encuentra en la tabla 5.16 es el valor *111010*, por lo tanto la categoría que le corresponde a este valor es *0x31*. Los valores de esta categoría son RRRR=3 y SSSS=1 de acuerdo a la tabla 5.10 (*sección 5.1.5*). El número de bits adicionales del código Huffman es SSSS=1, entonces el bit que sigue al código Huffman es un 0. El valor de 0 indica que el coeficiente AC es negativo, así que el valor 0 se complementa a uno, por lo tanto el coeficiente AC diferente de cero es igual a -1. Como el valor RRRR es igual 3, al coeficiente AC le preceden tres valores iguales a 0. Así que el arreglo zigzag_rec queda de la siguiente forma:

$$zigzag\_rec[64] = [35, 0, -2, 0, 0, 0, -1, ...] \quad (5.17)$$

Finalmente, la cadena de bits que queda por codificar es el valor 1010 el cual corresponde al símbolo de fin de bloque EOB de la figura 5.10, por lo que el resto de los valores del arreglo zigzag_rec son iguales a 0. Así, el arreglo zigzag_rec queda:

zigzag_rec[64] = [35, 0, -2, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] (5.18)

En el caso raro en el cual el 63º coeficiente es distinto de cero, el fin de bloque no se codifica. También se puede dar el caso en el que se presente una *longitud sucesiva de ceros (ZRL) (sección 3.4.4)*. Este caso se presenta cuando la longitud de ceros es mayor a 15, lo que se interpreta como una longitud de 15 ceros seguida por un coeficiente de amplitud cero.

Ordenamiento zigzag inverso

Una vez obtenido el arreglo zigzag_rec, se reordenan los elementos del arreglo en una matriz de coeficientes para la decuantización de datos. Los algoritmos para realizar este ordenamiento son los mismos que los de las figuras 5.6 y 5.8, solo que ahora los datos se leen del arreglo zigzag_rec y se almacenan en la matriz de coeficientes cuantizados, como se muestra en la tabla 5.17.

35	0	0	-1	0	0	0	0
-2	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Tabla 5.17 Matriz de coeficientes DCT listos para la decuantización.

5.2.2 Proceso de Decuantización

Una vez que se tiene la matriz de coeficientes DCT cuantizados, se procede aplicar la decuantización. El proceso para la decuantización se obtiene despejando el valor Sq_{uv} de la ecuación 5.11, con lo que se obtiene la ecuación 5.19:

$$S_{uv} = \left(\frac{Sq_{uv}}{Q_{uv}^{-1} \bullet QF} \right) \quad (5.19)$$

Como las divisiones son muy pesadas computacionalmente, modificamos la ecuación 5.19, y obtenemos la ecuación 5.20, la cual usa solo multiplicaciones.

$$S_{uv} = Sq_{uv} \bullet Q_{uv} \bullet QF^{-1} \qquad (5.20)$$

donde:

S_{uv} es el *valor del coeficiente recuperado*
Sq_{uv} es el *valor del coeficiente cuantizado*
Q_{uv} es el es el *valor de la tabla de cuantización original*
QF^{-1} es el *inverso del factor de cuantización*

Cabe resaltar que en este caso no es necesario utilizar las tablas de cuantización inversas 5.5 y 5.6, sino que se utilizan las tablas originales de cuantización, mostradas en las tablas 5.18 y 5.19.

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 5.18 Tabla de cuantización original para 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 5.19 Tabla de cuantización original para crominancias.

La ecuación 5.20 se aplica a los datos de la tabla 5.17, para obtener los valores de los coeficientes de la DCT decuantizados. Si comparamos las tablas 5.4 y 5.20, podemos observar que al cuantizar los datos, los valores originales ya no pueden recuperarse de forma exacta por lo que la cuantización es un proceso con pérdidas (*sección 2.5*).

280.0	0.0	0.0	-8.0	0.0	0.0	0.0	0.0
-12.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tabla 5.20 Matriz de coeficientes decuantizados de la DCT.

5.2.3 Algoritmo optimizado de la IDCT

Una vez que se obtiene la matriz de coeficientes decuantizados, se procede a aplicar la *transformada inversa de la DCT (IDCT)* en dos dimensiones. El procedimiento para obtener la IDCT bidimensional es muy similar al que realizamos para el cálculo de la DCT. En primer lugar, aplicamos el algoritmo de la IDCT unidimensional propuesto por Wang, Suehiro y Hatori de la figura 5.10 a las filas del bloque de 8x8 para obtener la matriz intermedia. Posteriormente, sobre la matriz obtenida aplicamos nuevamente la transformada unidimensional pero ahora sobre las columnas para obtener la matriz de coeficientes de la IDCT bidimensional [20].

En la figura 5.10, los datos de entrada son los coeficientes de la DCT *X(0)* a *X(7)*, mientras que las variables *x(0)* a *x(7)* representan los coeficientes de la IDCT, es decir, los valores de la MCU reconstruida desplazada en nivel. El procedimiento para calcular los coeficientes de la IDCT es muy similar al de la sección 5.1.3, lo único que cambia es la dirección del flujo de la información. El algoritmo completo para el cálculo de la IDCT unidimensional se representa por el conjunto de ecuaciones 5.21 y 5.22.

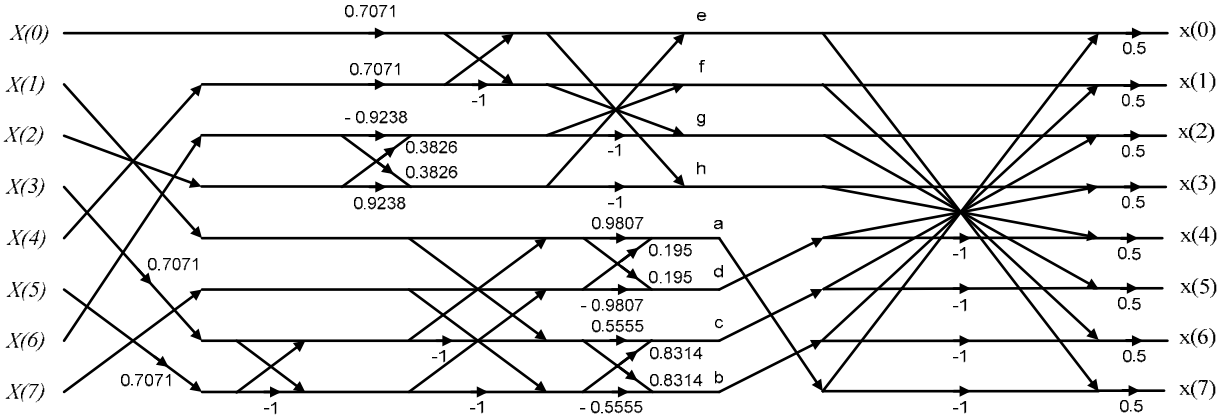


Figura 5.10 Algoritmo para el cálculo de la IDCT propuesto por Wang, y Suehiro y Hatori.

$$
\begin{aligned}
a &= 0.4904*(X[1] + 0.7071*X[3] + 0.7071*X[5]) + 0.0975*(X[7] + 0.7071*X[3] - 0.7071*X[5]) \\
b &= 0.4157*(X[1] - 0.7071*X[3] - 0.7071*X[5]) - 0.2778*(X[7] - 0.7071*X[3] + 0.7071*X[5]) \\
c &= 0.2778*(X[1] - 0.7071*X[3] - 0.7071*X[5]) + 0.4157*(X[7] - 0.7071*X[3] + 0.7071*X[5]) \\
d &= 0.0975*(X[1] + 0.7071*X[3] + 0.7071*X[5]) - 0.4904*(X[7] + 0.7071*X[3] - 0.7071*X[5]) \\
e &= (0.4619*X[2] + 0.1913*X[6] + 0.3536*X[0] + 0.3536*X[4]) \\
f &= (0.1913*X[2] - 0.4619*X[6] + 0.3536*X[0] - 0.3536*X[4]) \\
g &= -(0.1913*X[2] + 0.4619*X[6] + 0.3536*X[0] - 0.3536*X[4]) \\
h &= -(0.4619*X[2] - 0.1913*X[6] + 0.3536*X[0] + 0.3536*X[4])
\end{aligned}
\qquad (5.21)
$$

$$
\begin{aligned}
x[0] &= a + e & \qquad x[4] &= h - d \\
x[1] &= b + f & \qquad x[5] &= g - c \\
x[2] &= c + g & \qquad x[6] &= f - b \\
x[3] &= d + h & \qquad x[7] &= e - a
\end{aligned}
\qquad (5.22)
$$

Para obtener los coeficientes de la IDCT bidimensional de los coeficientes de la tabla 5.15, primero se toman los elementos de la primera fila como muestras de entrada, con lo que se obtienen los coeficientes IDCT unidimensionales, que corresponden a la primera fila de la matriz intermedia. Posteriormente, se aplicar la IDCT unidimensional a las siguientes filas de la matriz, con lo que se obtiene la matriz intermedia procesada por filas como se muestra en la tabla 5.21.

95.68237	99.78808	102.931	101.2306	96.78545	95.08501	98.22793	102.3336
-4.2432	-4.2432	-4.2432	-4.2432	-4.2432	-4.2432	-4.2432	-4.2432
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Tabla 5.21 Matriz intermedia de la IDCT bidimensional procesada por filas.

Posteriormente, aplicamos la IDCT tomando como muestras de entrada los elementos de la primera columna de la matriz intermedia. Después de que se aplica la IDCT, los datos de salida corresponden a la primera columna de la matriz final de coeficientes IDCT, este proceso continúa hasta que se procesan todas las columnas de la matriz intermedia. El resultado es la matriz final de coeficientes IDCT bidimensional de la tabla 5.22.

32.0	33.0	34.0	34.0	32.0	32.0	33.0	34.0
32.0	34.0	35.0	34.0	32.0	32.0	33.0	34.0
33.0	34.0	35.0	35.0	33.0	32.0	34.0	35.0
33.0	35.0	36.0	35.0	34.0	33.0	34.0	36.0
34.0	36.0	37.0	36.0	35.0	34.0	35.0	37.0
35.0	36.0	38.0	37.0	35.0	35.0	36.0	37.0
36.0	37.0	38.0	38.0	36.0	35.0	36.0	38.0
36.0	37.0	38.0	38.0	36.0	36.0	37.0	38.0

Tabla 5.22 Matriz final de la IDCT bidimensional.

5.2.4 Incremento de nivel

Como la unidad de datos MCU reconstruida de la tabla 5.22 está desplazada en nivel, se incrementa su nivel sumando 128 a cada elemento de los datos como se ilustra en la tabla 5.23, donde la unidad de datos obtenida es la unidad de datos MCU reconstruida y difiere un poco de la MCU original de la tabla 5.1. Los análisis del error y de calidad del video se realizan en la *sección 6*.

160	161	162	162	160	160	161	162
160	162	163	162	160	160	161	162
161	162	163	163	161	160	162	163
161	163	164	163	162	161	162	164
162	164	165	164	163	162	163	165
163	164	166	165	163	163	164	165
164	165	166	166	164	163	164	166
164	165	166	166	164	164	165	166

Tabla 5.23 Unidad MCU después del incremento de nivel

Posteriormente, las unidades de datos se ordenan en memoria como se indicó en la sección 5.1.1. Una vez que se han obtenido todas las unidades MCU de los tres componentes de la imagen se procede a la visualización de video, que se describe en la siguiente sección. Finalmente, el proceso de descompresión se repite hasta que se decodifican todas las imágenes de la secuencia de video.

5.2.5 Visualización del video reconstruido en la PC

Una vez que se tiene una secuencia de video reconstruida, es necesario visualizarla para evaluar su calidad. Para esto, los datos del video reconstruido se guardan en un archivo, el cual es leído por el programa de visualización de video de la PC. Debido a que los cuadros de video reconstruido están en formato YCrCb 4:2:0, éstos deben ser transformados al espacio RGB para poder ser visualizados en pantalla *(sección 1.3.3)*. Para pasar al espacio RGB, el primer paso es pasar la información de un submuestreo 4:2:0 a un 4:4:4. Esta transformación se realiza a través de una interpolación lineal entre las muestras disponibles Cb y las muestras disponibles Cr para encontrar el valor de las muestras que faltan. La interpolación primero se realiza sobre las columnas del componente, con lo que tenemos un submuestreo 4:2:2. Posteriormente, la interpolación se realiza sobre las filas de las muestras del componente, con lo que se obtiene el formato 4:4:4. Este procedimiento se ilustra en la figura 5.11.

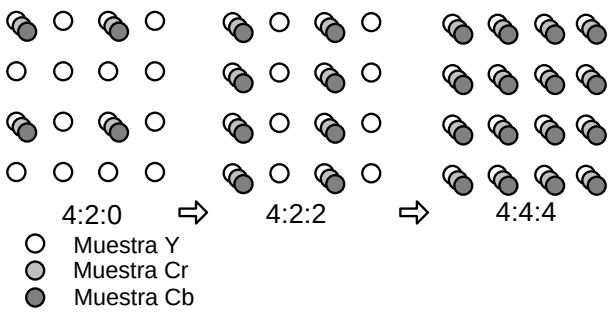


Figura 5.11 Conversión de formato de submuestreo 4:2:0 a 4:4:4

Para desplegar la secuencia de video sobre la pantalla de la PC, los datos con formato YCrCb 4:4:4 se transforman del espacio de color RGB. Una vez que los datos de la imagen están en RGB, el programa despliega un mapa de bits de la imagen en la ventana principal del programa de despliegue. Este proceso se repite, pero ahora se procesan los datos del buffer Pong, mientras se llena el buffer Ping.

5.3 Transmisión de video por el puerto serie

Como se mencionó en la *sección 4.1*, una de las expectativas de este trabajo a futuro es que el video comprimido pueda ser transferido a una PC o a algún otro dispositivo para su visualización en tiempo real. En este proyecto se implementó la transmisión de video por el puerto serie con la finalidad de evaluar su factibilidad de ser integrado en el sistema en un futuro. En esta etapa, se realizaron dos pruebas principales: en la primera, una vez que se adquiere y adecua una imagen de video sin comprimir a la memoria externa del DSP, ésta se transmite por el puerto serie y se visualiza en una PC. En la segunda prueba, una secuencia de video descomprimido almacenado en la memoria externa del DSP es transmitida por el puerto serie y desplegada en una PC.

En este proyecto, la descompresión de video MJPEG se realizó utilizando un DSP, pero en un proyecto a futuro la descompresión debe ser ejecutada por una PC, como se mostró en la figura 4.2 de la *sección 4.1*. Al transmitir datos de video por el puerto serie, también es posible transmitir datos de video comprimido por el puerto serie, y de acuerdo a la figura 4.2 solo faltaría adaptar la descompresión de video MJPEG para que la realice una PC. Esta adaptación quedó fuera del alcance de este proyecto debido a la complejidad de migrar algunas funciones fundamentales de la descompresión implementadas en el DSP en lenguaje C al lenguaje C#, el cual utilizamos en la visualización de video en una PC.

En una transmisión serial, el dispositivo transmisor envía los bits uno a la vez en forma secuencial a través de una interface. Tanto el dispositivo transmisor como el receptor requieren una señal de reloj para decidir cuándo enviar un bit o cuando leer cada bit. Existen dos tipos de formatos de transmisión serial, la síncrona y la asíncrona, las cuales se describen enseguida.

Formato Síncrono: En una transmisión síncrona, todos los dispositivos utilizan una señal de reloj común generado por uno de los dispositivos o por una fuente externa. Todos los bits transmitidos son sincronizados por el reloj.

Formato Asíncrono: En una transmisión asíncrona, no se requiere una línea de reloj, ya que cada dispositivo tiene su propio reloj. Cada byte transmitido incluye un *bit de Inicio* para sincronizar los relojes, y uno o más *bits de Parada* para indicar el fin de la transmisión del byte de datos. El puerto serial RS-232 en las PCs utiliza el formato de comunicación asíncrona para comunicación con módems u otros dispositivos.

Una transmisión asíncrona puede utilizar varios formatos de datos. El formato utilizado en este proyecto es el formato *8-N-2*, donde el transmisor envía cada byte de datos con un bit de inicio, seguido por los 8 bits de datos, iniciando con el bit menos significativo, para finalizar con 2 bits de parada. La *N* indica que la transmisión no utiliza bit de paridad para detección de error. La tasa de bits es el número de bits por segundo transmitidos o recibidos por unidad de tiempo, y usualmente es expresado como *bits por segundo (bps)*, la tasa de bits empleada en este proyecto es de *115,200 bps*.

Las PCs y muchos microcontroladores utilizan un dispositivo llamado *Transmisor Receptor Universal Asíncrono (UART)*, el cual desempeña las funciones de transmisión y recepción de datos. En las PCs, los sistemas operativos y los lenguajes de programación incluyen herramientas para la programación del puerto serial, de tal forma que la arquitectura de la UART es transparente para el usuario.

Para configurar una transmisión serial, la aplicación selecciona una tasa de datos junto con otros parámetros, y habilita la comunicación del puerto deseado. Para enviar un byte, la aplicación escribe el byte al buffer transmisor del puerto seleccionado, y la UART envía el dato, bit por bit, en el formato requerido, adicionando los bits de inicio, parada y los bits de paridad necesarios. De forma similar, los bytes recibidos son almacenados automáticamente en un buffer. La UART puede activar una interrupción para notificar al CPU, y a su vez a la aplicación de los datos que están llegando y de otros eventos [28].

Puerto serie de la DSKcam

El puerto serie de la tarjeta DSKcam está basado en la UART TI16C550. Este dispositivo puede operar hasta 1 Mb/s en comunicación asíncrona full duplex. Para la configuración de la UART, utilizamos un driver de dispositivo tipo *IOM (Mini-driver de entrada/salida)*. La configuración del conector tipo D de 9 pines para el puerto serie RS-232 se muestra en la tabla 5.24. La interface serial de la DSKcam se conecta a la PC utilizando una conexión estándar tipo D. La tasa máxima de datos permitida en los PCs es de 115,200 bps [22].

PIN	RS232	Descripción
1		
2	Tx	Señal de transmission
3	Rx	Señal de recepción
4		
5	GND	Tierra
6		
7	CTS	Listo para recibir
8	RTS	Petición de envío
9		

Tabla 5.24 Configuración de los pines del puerto serial de la tarjeta DSKcam.

Transmisión por el puerto serie de la DSKcam a la PC

Para la transmisión del puerto serie, se utiliza un mini driver IOM para la configuración de la UART. A través de este driver, se puede empezar la transmisión de datos desde el DSP a la PC. Los parámetros para configurar el puerto serial de la DSKcam son los siguientes:

- Palabra de 8 bits
- Sin paridad
- 2 bits de STOP
- Tasa de datos 115,200

En el otro lado, la PC utiliza el programa que se desarrolló con Visual C# para leer los datos de video (*sección 5.2.5*). Como una de las expectativas de este trabajo a futuro es que el video comprimido pueda ser transferido a una PC o a algún otro dispositivo para su visualización en tiempo real, se tiene planeado que el DSP transmita los datos de video a otro dispositivo receptor (consideraremos una PC), y que el dispositivo receptor realice las tareas de descompresión, conversión de espacio de color y visualización de video.

El planteamiento del sistema es que cuando la información del video comprimido sea transferida a la computadora, los datos se almacenen en un buffer tipo Ping-Pong, el cual está compuesto por dos buffers. Una vez que uno de los dos buffers está lleno con los datos necesarios para reconstruir una imagen de la secuencia de video, este buffer, al que llamaremos Ping, será procesado mientras que los datos comprimidos del siguiente cuadro se almacenan en el otro buffer (buffer Pong). Cuando el buffer Pong está lleno y la imagen del buffer Ping ya ha sido desplegada, se procede a descomprimir la imagen del buffer Pong y los datos provenientes del puerto serie se almacenan en el buffer Ping, y el ciclo se repite nuevamente.

El diagrama de la parte de transmisión del puerto serie de la tarjeta DSKcam se mostró en la figura 4.6 de la *sección 4.4*. La figura 5.12 muestra un diagrama a bloques propuesto del proceso de recepción por el puerto serie y la visualización del video en la PC. Cabe recordar que en este proyecto para efectos de pruebas, la descompresión se realiza con el DSP, y en un futuro se planea implementar esta etapa en una PC, así que en el caso de transmisión del video reconstruido o sin comprimir, en el sistema de recepción de la figura 5.12 se omite el proceso de descompresión.

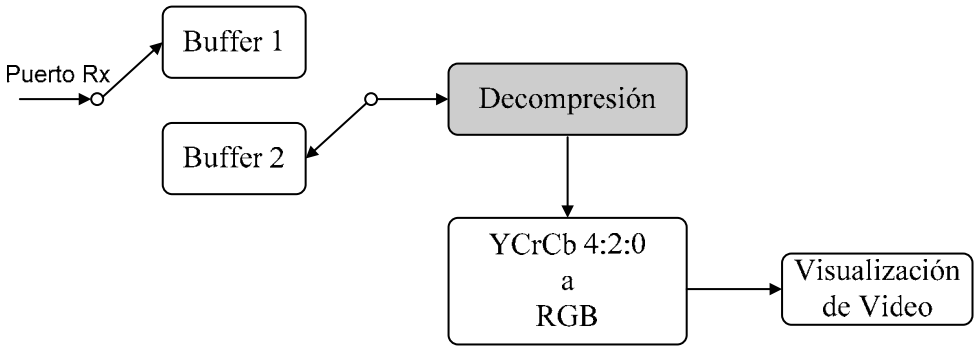


Figura 5.12 Esquema de recepción de datos propuesto para la visualización de video.

5.4 Sincronización de tareas del sistema

Muchas aplicaciones de procesamiento digital de señales deben ejecutar varias funciones al mismo tiempo, frecuentemente en respuesta a eventos externos tales como la disponibilidad de datos o la presencia de una señal de control. Estas funciones son muy importantes en el sistema y son llamadas *hilos*. Dentro del sistema operativo DSP/BIOS soportado por el DSP, el término *hilo* es utilizado para incluir un conjunto independiente de instrucciones ejecutadas por el DSP. Un hilo es un punto único de control que puede contener una subrutina, una ISR, o una llamada a una función.

5.4.1 Tipos de hilos soportados por el DSP/BIOS

El DSP/BIOS proporciona soporte para varios tipos de hilos con diferentes prioridades de ejecución. Los tipos de hilos ordenados de la prioridad más alta a la más baja son [29]:

- **Interrupciones por Hardware (HWI):** Estas interrupciones son activadas en respuesta a eventos asíncronos externos que ocurren en el ambiente del DSP. Una función HWI (también llamada rutina de servicio de interrupción o ISR) es ejecutada después de que es activada una interrupción por hardware para desempeñar una tarea crítica. Las funciones HWI son los hilos con la más alta prioridad en una aplicación con el DSP/BIOS.

- **Interrupciones por Software (SWI):** Mientras las ISRs son activadas por una interrupción por hardware, las interrupciones por software son activadas por medio de llamadas a las funciones SWI dentro del programa. Las interrupciones por software proporcionan niveles de prioridad adicionales entre las interrupciones por hardware y el hilo de fondo. Al igual que las HWIs, los hilos SWI nunca suspenden su ejecución hasta que terminan de realizar su función.

- **Tareas (TSK):** Las tareas tienen una prioridad más alta que el hilo de fondo y una prioridad más baja que las interrupciones por software. Las tareas difieren de las interrupciones por software en que las primeras pueden ser suspendidas durante una ejecución hasta que haya suficientes recursos disponibles. DSP/BIOS proporciona varias estructuras que pueden ser utilizadas para comunicación inter-tareas y sincronización. Estas estructuras incluyen semáforos, colas y mailboxes.

- **Hilo de fondo:** Ejecuta el *loop de desocupado (IDL*) en la prioridad más baja en el DSP/BIOS. El loop IDL se ejecuta continuamente excepto cuando el control es apropiado por un hilo con una prioridad más alta.

El tipo y nivel de prioridad elegido para un hilo en un programa de aplicación influye en los tiempos de programación de las tareas y en su correcta ejecución. A continuación se presenta una comparación entre las *HWIs vs SWIs o Tareas* y las *SWIs vs Tareas*.

- **SWI o Tareas *vs* HWI**. El procesamiento crítico se debe desempeñar dentro de las rutinas de servicio de interrupción por hardware. Las funciones HWI pueden activar interrupciones por software o tareas para desempeñar procesamiento de bajo nivel. El uso de hilos de baja prioridad minimiza la cantidad del tiempo en el que las interrupciones están deshabilitadas, permitiendo que puedan ocurrir otras interrupciones.

- **SWI *vs* Tareas:** Las interrupciones por software se utilizan si las funciones tienen interdependencias y requerimientos de datos compartidos relativamente simples. Por el contrario, si los requerimientos son más complejos se deben utilizar las tareas. Mientras los hilos con prioridad más alta pueden apropiarse de hilos de prioridad más baja, únicamente las tareas pueden ser suspendidas para esperar otro evento, como la disponibilidad de recursos. Las tareas tienen más opciones que las SWIs cuando se utilizan datos compartidos. Todas las entradas necesarias para una función de interrupción por software pueden estar listas cuando el programa activa la SWI.

Dado que este proyecto requiere de recursos compartidos y los procesos presentan cierta complejidad, se decidió utilizar las tareas para la sincronización de procesos del sistema en el DSP. A continuación se describen las principales características de las tareas.

Tareas

Las tareas en el DSP/BIOS son hilos que son manejados a través del *módulo TSK*. El módulo TSK se utiliza para calendarizar dinámicamente el orden de ejecución de las tareas, basándose en el nivel de prioridad de cada tarea y en el estado de ejecución de la tarea actual. Esto asegura que el procesador sea utilizado por el hilo con mayor prioridad que esté listo para ejecutarse. Existen 15 niveles de prioridad disponibles para las tareas. El nivel de prioridad más bajo (0) está reservado para la ejecución del *loop idle*.

Programación y estados de ejecución de las tareas.

Cada objeto de tarea TSK puede estar en uno de los cuatro estados posibles de ejecución:

1) **Ejecución**, significa que la tarea es la única que se está ejecutando realmente sobre el procesador del sistema.

2) **Lista**, significa que la tarea está programada para su ejecución, sujeta a la disponibilidad del procesador.

3) **Bloqueada**, significa que la tarea no puede ejecutarse hasta que ocurra un evento particular dentro del sistema.

4) **Terminada**, indica que la tarea ha sido finalizada.

Las tareas son programadas para su ejecución de acuerdo con los niveles de prioridad asignados a la aplicación. No puede haber más de una tarea en ejecución. Como regla, ninguna tarea *lista* tiene un nivel de prioridad mayor que el de la tarea que se está ejecutando. A diferencia de muchos sistemas operativos de tiempo compartido que dan a cada tarea el tiempo del procesador que le corresponde, cada vez que una tarea de más alta prioridad está lista para ejecutarse (*lista*), el DSP/BIOS desplaza inmediatamente a la tarea actual y atiende a la tarea con prioridad más alta.

El máximo nivel de prioridad es *TSK_MAXPRI(15)*; el mínimo nivel de prioridad es *TSK_MINPRI(1)*. El nivel de prioridad 0 corresponde a *TSK_idle*, la cual es la tarea que ejecuta el *loop idle* o *background*. Si la prioridad es menor que 0, la tarea tiene prohibida su ejecución hasta que su prioridad sea elevada por otra tarea en un tiempo posterior. Si la prioridad es igual a *TSK_MAXPRI*, la ejecución de la tarea bloquea todas las otras actividades del programa, excepto el manejo de interrupciones por software o hardware.

Durante el curso de un programa, el modo de ejecución de cada tarea puede cambiar por varias razones. La figura 5.13 muestra como pueden cambiar los modos de ejecución. Las funciones en TSK, los semáforos y los módulos *SIO (streaming Input Output)* alteran el estado de ejecución de las tareas como: bloqueo o finalización de la tarea que está corriendo actualmente, poner en estado *lista* una tarea suspendida previamente, reprogramación de la tarea actual, etc. [29].

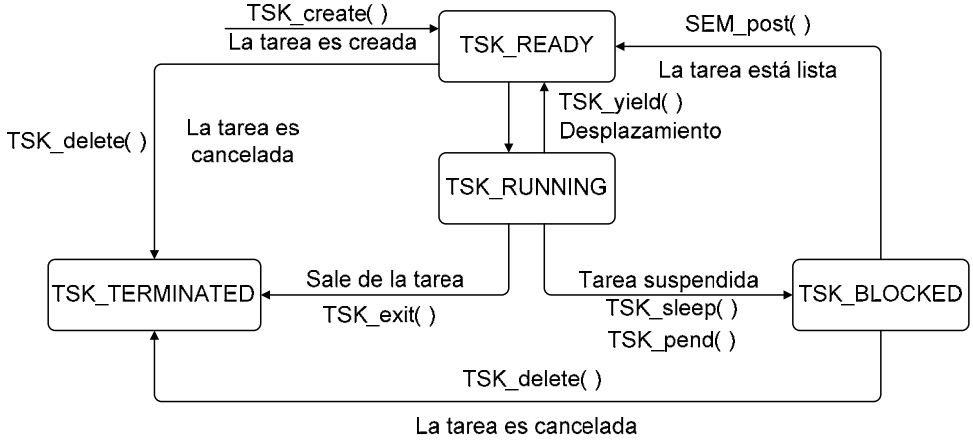


Figura 5.13 Transición de los diferentes estados de ejecución de una tarea.

En la figura 5.13 se observa que solo una tarea está en el modo TSK_RUNNING. Si todas las tareas del programa están bloqueadas y no ocurre ninguna interrupción por software o hardware, el DSP ejecuta la tarea *TSK_idle*, cuya prioridad es la más baja en el sistema.

Cuando la tarea TSK_RUNNING pasa a cualquiera de los otros tres estados, el control cambia el estado de la tarea con la prioridad más alta del estado LISTA (modo TSK_READY) al estado de EJECUCION. Una tarea TSK_RUNNING cambia de estado en alguna de las siguientes formas:

1) La tarea que está corriendo se pone en TSK_TERMINATED llamando a TSK_exit( ), la cual es llamada automáticamente en caso de que una tarea retorne de su función de más alto nivel. Después de que todas las tareas han retornado, el módulo TSK termina la ejecución del programa llamando a SYS_exit( ) con un código de estado 0.

2) La tarea en estado de EJECUCION pasa a TSK_BLOCKED cuando se llama a una función, por ejemplo SEM_pend( ) o TSK_sleep( ), esto ocasiona que la tarea actual suspenda su ejecución. Las tareas pueden pasar a este estado cuando están ejecutando ciertas operaciones de entrada/salida, están esperando la disponibilidad de algunos recursos compartidos o están inactivas (idle).

3) La tarea que se está ejecutando pasa a un estado TSK_READY cada vez que otra tarea con prioridad más alta esta lista para correr. TSK_setpri( ) puede causar este tipo de transición si la prioridad de la tarea actual no es mayor que la prioridad más alta en el sistema. Una tarea también puede utilizar TSK_yield( ) para ceder a otras tareas con la misma prioridad. Una tarea que cede el control está lista para ejecutarse.

Una tarea que está actualmente en estado TSK_BLOCKED pasa al estado LISTA en respuesta a un evento particular: terminación de una operación entrada/salida, disponibilidad de recursos compartidos, el transcurso de un periodo específico de tiempo, y así sucesivamente. En virtud de que la tarea pasa a un estado TSK_READY, dicha tarea es programada para su ejecución de acuerdo a su nivel de prioridad, y por supuesto, si su prioridad es mayor que la de la tarea que se está ejecutando actualmente, la tarea pasa inmediatamente al estado de ejecución. TSK programa las tareas de igual prioridad por orden de llegada [29].

5.4.2 Diseño del software para la sincronización de tareas

Para sincronizar los procesos del sistema, se decidió utilizar dos tareas. La primera tarea llamada *Cap_Comp( )*, realiza la captura y compresión de video MJPEG y es la tarea con mayor prioridad (prioridad 2). La segunda tarea realiza la transmisión de los datos por el puerto serie y tiene un nivel de prioridad 1. Esta tarea recibe el nombre de *echo ( )*. El nivel de prioridad de las tareas se configura a través del DSP/BIOS. Cabe aclarar que como una fase de prueba, en nuestro sistema transmitimos los datos de video sin comprimir, dado que la descompresión de los datos comprimidos por la PC se tiene planeado como una etapa a desarrollarse a futuro, sin embargo, nuestro sistema es capaz de transmitir video comprimido sin ningún problema, por lo que la tarea *Cap_Comp( )* es capaz de realizar la captura y compresión de video o bien solo la adquisición.

Para realizar la sincronización de tareas se utilizó un semáforo, debido a que los semáforos pueden coordinar el acceso a recursos compartidos entre un conjunto de tareas competentes (*sección 4.2.4*). El funcionamiento del programa de captura, compresión y transmisión de video por el DSP se ilustra en la figura 5.14 y funciona de la siguiente manera:

1) Al inicio, el programa se encuentra en el *background*, el cual es el hilo de más baja prioridad en el DSP/BIOS, como se muestra en la figura 5.14. El background ejecuta el loop de desocupado continuamente hasta que el control es asumido por otro hilo.

2) Posteriormente, nuestro sistema ejecuta la tarea con mayor prioridad, la tarea de captura y compresión *tsk Cap_Comp( )*, la cual entra a un loop infinito *while(1==1)*. A continuación, la tarea se queda en estado de espera TSK_BLOCKED poniendo al semáforo en *SEM_pend(&sem_prin, SYS_FOREVER)*, donde *sem_prin* es el nombre del semáforo principal que coordina las tareas, y *SYS_FOREVER* indica que el semáforo esperará indefinidamente hasta que sea liberado por la función *SEM_post( )*.

3) Al estar suspendida la tarea *Cap_Comp( )* se ejecuta la tarea *echo( )*, la cual entra a un loop infinito *while(1==1)* e inmediatamente libera a la tarea *captura_compresión* a través de la función *SEM_post(&sem_prin)*.

4) Cuando se libera la tarea *Cap_Comp( )*, ésta pasa al estado TSK_READY y enseguida al estado TSK_RUNNING, ejecutando la captura y compresión de video.

5) Dentro de la tarea *Cap_Comp( )* ocurre la interrupción externa por hardware (*VSync*) para realizar la transferencia por EDMA de las imágenes de video desde el sensor a la memoria externa del DSP.

6) Una vez que la tarea *Cap_Comp( )* termina de ejecutarse, pasa al estado TSK_BLOCKED, y la tarea *echo( )* pasa al estado TSK_RUNNING iniciando la transmisión serial de los datos comprimidos. Es importante mencionar que la tarea *echo( )* inicia la transmisión de datos y regresa el control del CPU a la tarea *Cap_Comp( )*, pero el puerto serial siguen transmitiendo datos de acuerdo a su configuración.

7) En el otro lado del sistema, la PC recibe los datos de video en formato YCrCb con submuestreo 4:2:0, los convierte a un submuestreo 4:4:4, los traslada al espacio de color R, G, B y posteriormente despliega las imágenes de video (*secciones 5.2.5 y 5.3*).

Una vez que se transmiten los datos, la tarea *echo( )* ejecuta la función SEM_Post liberando a la tarea *Cap_Comp( )* y el ciclo se vuelve a repetir. En el capítulo 6 se hace el análisis del funcionamiento tipo pipeline hardware-software del sistema.

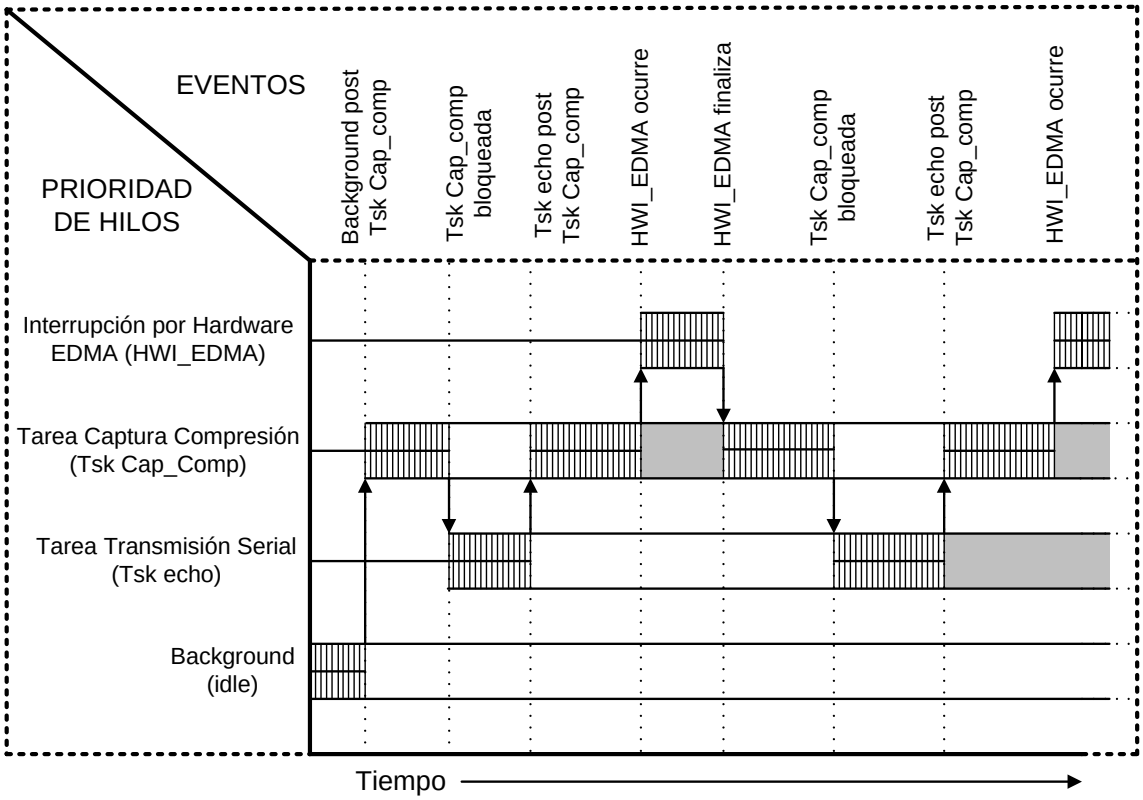


Figura 5.14 Ejecución de los hilos del sistema de captura y transmisión de video implementado.

Cabe hacer notar que cuando se utilizan tareas en el DSP/BIOS, la función *main* solo se utiliza para configurar parámetros y condiciones iniciales, posteriormente ejecuta al *loop idle* y el control es asumido por el DSP/BIOS a través de la calendarización de tareas, las cuales se sincronizan mediante semáforos, como se mencionó anteriormente. El código empleado para sincronizar las tareas se muestra en el anexo B.

RESUMEN

En este capítulo se explicó como se implementó la compresión de video MJPEG en el DSP, describiendo cada una de las etapas de la compresión, desde el incremento de nivel hasta la codificación entrópica. Posteriormente, se describieron cada una de las etapas del proceso de descompresión ejecutado por el DSP, al igual que el proceso para visualizar el video reconstruido en la PC. Además, se mencionó como se configuran los puertos tanto de la DSKcam como de la PC para realizar la transmisión serial de los datos de video desde la memoria externa del DSP a la PC. Al final del capítulo, se describió la implementación de la sincronización de las tareas del sistema utilizando un semáforo. En el siguiente capítulo se presenta el análisis de los resultados obtenidos en este trabajo.

6.- EVALUACION DE LOS RESULTADOS OBTENIDOS

En este capítulo se presentan los resultados obtenidos al adquirir varias secuencias de video en tamaño QVGA, VGA y QCIF, en el espacio de color YCrCb con diferentes formatos de submuestreo, así como el formato QCIF en blanco y negro. Además se muestran los resultados al aplicar los algoritmos de Compresión y Descompresión MJPEG sobre una secuencia de video capturada. 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 Adquisición y Compresión de video, y los errores obtenidos al recuperar las secuencias de video. Igualmente, se hace un análisis del pipeline hardware-software del sistema de adquisición y compresión de video MJPEG.

6.1 Evaluación de resultados de la adquisición de video

En la etapa de adquisición de video se logró capturar secuencias de video con diferente resolución y diferentes formatos de submuestreo. En el análisis de resultados se presentan los tiempos de transferencia de cada imagen desde el sensor de video a la memoria externa SDRAM del DSP, el tiempo que tarda el DSP en ordenar la imagen en sus componentes por separado y en algunos casos en cambiar la resolución y formato de submuestreo. Además se hace un análisis del espacio de memoria requerido por las secuencias de video de diferentes resoluciones y se muestran los resultados gráficos de la adquisición de varias secuencias de video.

En la tabla 6.1 se hace el análisis de los tiempos de procesamiento para una sola imagen de los diferentes formatos de video utilizados en el proyecto. Los resultados se obtuvieron utilizando el ambiente Code Composer Studio, estos tiempos se obtienen considerando que el DSP C6416 trabaja a 8000 *millones de instrucciones por segundo (MIPS)* a una frecuencia de 1 GHz [10].

Formato de video	Tiempo de transferencia por EDMA [ms]	Tiempo de adecuación de la imagen [ms]	Espacio en memoria [bytes]
VGA 4:2:2	23.734	429.5361	614400
QVGA 4:2:2	10.8591	43.1069	153600
QCIF 4:2:2	10.8591	42.8036	50688
QCIF 4:2:0	10.8591	40.1867	38016
QCIF Blanco y Negro	10.8591	8.7694	25344

Tabla 6.1 Tiempos de procesamiento y espacio en memoria de una imagen de video.

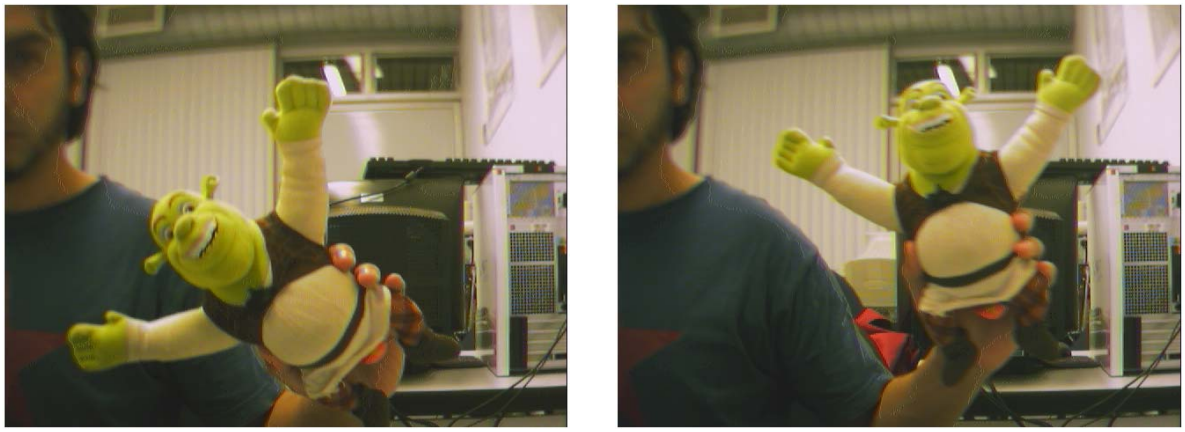
En la tabla 6.1 se puede observar que el tiempo de transferencia por EDMA es el mismo para los formatos de video QVGA y QCIF, esto se debe a que el sensor de video solo puede adquirir imágenes con resolución VGA o QVGA, en el caso del video en formato QCIF 4:2:0 se adquirió el video con formato QVGA con submuestreo 4:2:2 y después se adaptó a los formatos QCIF 4:2:2 y 4:2:0 respectivamente *(sección 4.6)*. El tiempo de adecuación de la imagen se refiere al tiempo que tarda el DSP en ordenar la imagen en sus componentes Y, Cr y Cb, ya que el sensor de video transfiere las imágenes a la memoria del DSP de forma entrelazada *(secciones 4.4.1 y 4.5.2)*. Además, en el caso del formato QCIF se hace una adecuación del tamaño de imagen y del submuestreo *(sección 4.6)*.

Para operar en tiempo real, el tiempo de procesamiento de la señal debe de ser menor al tiempo de muestreo. En nuestro caso, consideramos cada imagen como muestra, y el tiempo entre la adquisición de cada imagen como el tiempo de muestreo. En el caso del tiempo de transferencia por EDMA para transmitir una imagen a color con formato QVGA 4:2:2, desde el sensor de video a la memoria externa del DSP, el cual es de 11 ms, no existe ningún problema para estar dentro del tiempo real, ya que si se utiliza una tasa de 29 fps (utilizada en estándares de video como NTSC), el tiempo entre cada cuadro será de 34 ms y la transferencia EDMA ocupa solo una tercera parte del tiempo de muestreo. Por esta razón, nuestro sistema puede adquirir video a color QVGA 4:2:2 en tiempo real a una tasa menor o igual a 29 fps.

Como se observa en la tabla 6.1, en el caso de imágenes a color la adecuación al formato QCIF 4:2:0 requiere menos tiempo de procesamiento y es el que requiere menos espacio de almacenamiento en memoria, lo que permite que se puedan almacenar mayor número de cuadros de video a color en la memoria externa del DSP.

Por otro lado, en algunos sistemas no es muy necesario que el video sea a color, por lo que el video en blanco y negro es suficiente, como en los sistemas de vigilancia, por ejemplo. En la tabla 6.1 se puede observar que el tiempo de adecuación de la imagen en blanco y negro se reduce hasta un 20% aproximadamente del tiempo de adecuación de una imagen con formato QCIF 4:2:0. Además, como las imágenes en blanco y negro solo utilizan el componente de luminancia Y, el espacio de almacenamiento se reducen casi a un 67% del espacio necesario para el formato a color QCIF 4:2:0, reduciéndose además la cantidad de datos a procesar.

En la figura 6.1 presentamos los resultados gráficos al adquirir una secuencia de video a color en tamaño VGA, con un formato de submuestreo 4:2:2



(a) **(b)**

Figura 6.1a Imágenes (a) inicial y (b) intermedia de una secuencia de video adquirida en tamaño VGA con formato de submuestreo 4:2:2

(c)

Figura 6.1b Imagen final de una secuencia de video adquirida en tamaño VGA con formato de submuestreo 4:2:2

La figura 6.2 muestra una secuencia de video a color en tamaño QVGA, con un formato de submuestreo 4:2:2.

(a) **(b)** **(c)**

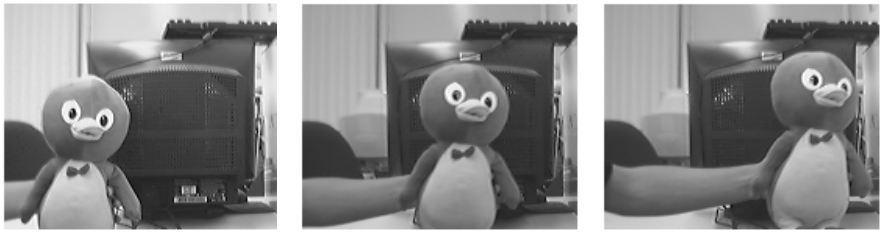
Figura 6.2 Imágenes (a) inicial, (b) intermedia y (c) final de una secuencia de video adquirida en tamaño QVGA con formato de submuestreo 4:2:2

En la figura 6.3 se observa la extracción de imágenes en tamaño QCIF, a partir de imágenes en tamaño QVGA, con un formato de submuestreo 4:2:0, este formato de video es el que utilizamos para evaluar la compresión MJPEG.

(a) **(b)** **(c)**

Figura 6.3 Imágenes adquiridas en tamaño QCIF con submuestreo 4:2:0

En la figura 6.4 se muestra una secuencia de video en blanco y negro (solo con componentes Y), este formato de video es una alternativa si se pretende implementar un sistema de vigilancia y se quiere aproximar al tiempo real.



(a) **(b)** **(c)**

Figura 6.4 Imágenes adquiridas en tamaño QCIF en blanco y negro (solo componente Y)

En las figuras 6.5 y 6.6 se presenta una comparación visual entre una imagen en formato QCIF con submuestreo 4:2:2 y una imagen QCIF con submuestreo 4:2:0. En estas figuras, se aprecia gráficamente que la información de la crominancia utilizando el submuestreo 4:2:0 se reduce a la mitad con respecto al submuestreo 4:2:2.

(a) **(b)** **(c)** **(d)**

Figura 6.5 Componentes de una imagen adquirida en tamaño QCIF 4:2:2
(a) Componente Y, (b) Componente Cr, (c) Componente Cb, (d) Imagen total

(a) **(b)** **(c)** **(d)**

Figura 6.6 Componentes de una imagen adquirida en tamaño QCIF 4:2:0
(a) Componente Y, (b) Componente Cr, (c) Componente Cb, (d) Imagen total

En las figuras 6.5 y 6.6 se puede observar que al momento de desplegar la imagen total a color, la calidad de la imagen no cambia significativamente al utilizar submuestreo 4:2:2 o 4:2.0. Por esta razón, en el proyecto se decidió utilizar el formato de video QCIF con un submuestreo 4:2:0, ya que se reduce significativamente la cantidad de información a procesar, y a transmitir o almacenar.

6.2.- Evaluación de resultados de la compresión y descompresión

Para poder hacer una evaluación de los diferentes niveles de compresión se adquirió una secuencia de video de 24 cuadros de tamaño QCIF a color con formato de submuestreo 4:2:0, y se le aplicó la compresión MJPEG. Para variar los niveles de compresión se utilizó un factor de cuantización definido como QF *(sección 5.1.4)*. Los valores de QF utilizados en este proyecto fueron 2, 0.5 y 0.2, debido a que estos factores proporcionan una calidad de video reconstruido muy buena, regular y mala. Al aumentar la compresión se reduce la cantidad de memoria necesaria para almacenar el video comprimido, pero la calidad del video reconstruido se reduce.

6.2.1 Factor de compresión y porcentaje de error del video reconstruido

Para evaluar el número de bytes que requiere el video comprimido, cuando se aplica la compresión JPEG a cada imagen, se utiliza un contador, el cual nos proporciona el número de palabras de datos que ocupan los datos del video comprimido en memoria. Como se mencionó en la *sección 5.1.5*, los datos codificados se almacenan en palabras de 16 bits, por lo que el total de bytes necesarios para almacenar el video comprimido está dado por el doble del valor del contador. Una vez que se tiene el total de bytes de la imagen comprimida, el *Factor de Compresión* está dado por la ecuación 6.1.

%Factor compresión = (No. Bytes Video Comprimido / No. Bytes Video Original) x 100% (6.1)

En la tabla 6.2 se presentan los factores de compresión obtenidos en la compresión de la imagen inicial de la secuencia de video a color de 24 cuadros de la figura 6.7, para diferentes valores QF, y en la tabla 6.3 se muestran los factores de compresión para toda la secuencia de video.

Factor de Cuantización	Espacio Original [Bytes]	Espacio Comprimido [Bytes]	Factor de Compresión %
QF = 2	38016	5328	14.01%
QF = 0.5	38016	2242	5.89%
QF = 0.2	38016	1248	3.28%

Tabla 6.2 Resultados de la compresión para la imagen inicial a color con diferente valor de QF.

Factor de Cuantización	Espacio Original [Bytes]	Espacio Comprimido [Bytes]	Factor de Compresión %
QF = 2	912384	111624	12.23%
QF = 0.5	912384	49026	5.37%
QF = 0.2	912384	14080	1.54%

Tabla 6.3 Resultados de la compresión de la secuencia de video a color con diferente valor QF.

La figura 6.7 muestra la secuencia de video original a color, y en las figuras 6.8 a la 6.10 se muestra la calidad visual de las secuencias de video reconstruidas con diferente valor QF.

Figura 6.7 Secuencia de video original a color.

Figura 6.8 Secuencia de video reconstruida utilizando un factor QF = 2.

Figura 6.9 Secuencia de video reconstruida utilizando un factor QF = 0.5.

Figura 6.10 Secuencia de video reconstruida utilizando un factor QF = 0.2.

Como se puede observar en las figuras 6.8 a 6.10, conforme el valor del factor QF es menor, la calidad del video reconstruido disminuye, de tal forma que para un valor de 0.2 la calidad visual es muy mala, esto se debe a que un factor de cuantización QF muy pequeño genera que los elementos de las matrices de cuantización tomen valores grandes con lo cual al momento de efectuar la cuantización, se pierden muchos de los coeficientes AC generados por la DCT, por lo que muy es difícil conservar los detalles espaciales finos.

En la figura 6.9 se puede apreciar que con un factor de cuantización QF=0.5, se tiene una calidad de video reconstruido aceptable, la cual es lo suficientemente adecuada para algunas aplicaciones, como sistemas de vigilancia o video conferencias, por ejemplo. Por esta razón, en el caso de la secuencia de video en blanco y negro se decidió hacer el análisis utilizando solo un factor de cuantización QF=0.5.

En la tabla 6.4 se presenta el factor de compresión obtenido en la compresión de la imagen inicial de la secuencia de video en blanco y negro de 24 cuadros de la figura 6.11, para un valor QF=0.5 y en la tabla 6.5 se muestra el factor de compresión para toda la secuencia de video.

Factor de Cuantización	Espacio Original [Bytes]	Espacio Comprimido [Bytes]	Factor de Compresión %
QF = 0.5	25344	1606	6.34%

Tabla 6.4 Resultados de la compresión para la imagen inicial a blanco y negro con QF=0.5.

Factor de Cuantización	Espacio Original [Bytes]	Espacio Comprimido [Bytes]	Factor de Compresión %
QF = 0.5	608256	33728	5.55%

Tabla 6.5 Resultados de la compresión de la secuencia de video a blanco y negro con QF=0.5.

En la figura 6.11 se muestra una secuencia de video en blanco y negro, y en la figura 6.12 se muestra la secuencia de video reconstruida utilizando un factor QF=0.5.

Figura 6.11 Secuencia de video original en blanco y negro.

Figura 6.12 Secuencia de video en blanco y negro reconstruida utilizando un factor QF = 0.5

Por otro lado, es necesario realizar un análisis matemático del error que presentan las imágenes del video reconstruidas con respecto a las imágenes de la secuencia de video original. El análisis realizado se basa en la ecuación 6.2 [16], la cual realiza la sumatoria total de la diferencia que se produce entre los respectivos elementos de la imagen original y la reconstruida. Los resultados obtenidos en el cálculo del error para las imágenes a color, y blanco y negro, se muestran en la tabla 6.6.

$$\%error = \left(\frac{1}{N} \sum_{i=1}^{N} \frac{\left| P_{original}(i) - P_{reconstruida}(i) \right|}{P_{original}(i)} \right) \text{x}100 \qquad (6.2)$$

	Factor de Cuantización	% error
Imagen a Color	QF = 2	1.52%
	QF = 0.5	2.85%
	QF = 0.2	5.26%
Imagen en Blanco y Negro	QF = 0.5	1.35%

Tabla 6.6 Resultados obtenidos para él % de error en las imágenes de video reconstruidas.

En la tabla 6.6 se puede observar que conforme el factor QF disminuye, el porcentaje de error se incrementa, lo cual se ve reflejado en la calidad visual de las secuencias reconstruidas (figuras 6.8 a 6.10), puesto que a mayor porcentaje de error, las secuencias de video reconstruidas presentan una menor calidad visual.

6.2.2 Tiempos de procesamiento para la compresión

Otro parámetro importante a evaluar es el tiempo de ejecución del DSP en el proceso de compresión. Los tiempos de ejecución se evaluaron para cada uno de los diferentes valores de QF que se aplicaron a la secuencia de video a color. La tabla 6.7 muestra los tiempos de compresión para la imagen inicial de la secuencia de video y la tabla 6.8 indica los tiempos para toda la secuencia.

Factor de Cuantización	Tiempo de Compresión [ms]
QF = 2	114
QF = 0.5	110
QF = 0.2	110

Tabla 6.7 Tiempo de compresión para la imagen inicial a color con diferente valor QF

Factor de Cuantización	Tiempo de Compresión [s]
QF = 2	2.731
QF = 0.5	2.645
QF = 0.2	2.315

Tabla 6.8 Tiempo de compresión para la secuencia de video a color de 24 cuadros con diferente valor QF

En las tablas 6.7 y 6.8 se puede observar que no hay mucha diferencia en los tiempos de compresión al utilizar diferentes valores QF, por lo tanto el factor QF no es un parámetro relevante para optimizar los tiempos de ejecución de la compresión.

La tabla 6.9 muestra los tiempos de compresión para la imagen inicial y para toda la secuencia de video en blanco y negro utilizando un valor QF = 0.5. En la tabla 6.9 se observa que el tiempo de compresión de una Imagen en blanco y negro se reduce en un 67% del tiempo requerido para comprimir una imagen a color.

Factor de Cuantización	Tiempo de Compresión de una Imagen [ms]	Tiempo de Compresión de la Secuencia [s]
QF = 0.5	74	1.764

Tabla 6.9 Tiempo de compresión para la imagen inicial y para toda la secuencia de video de 24 cuadros en blanco y negro con QF=0.5.

Por otro lado, dado que el método de compresión MJPEG se basa en procesar unidades de datos de 8x8 pixeles, se hizo una evaluación del tiempo de ejecución de cada una de las etapas de compresión que se aplican a las unidades de datos. El objetivo de conocer los tiempos de ejecución de las diferentes etapas es determinar cuáles son las etapas que requieren mayor tiempo de procesamiento y en un futuro puedan ser optimizadas, para que se pueda llegar al tiempo real.

Las etapas evaluadas para el algoritmo de compresión son: obtención de unidades de datos y desplazamiento de nivel, DCT, Cuantización, Zigzag y DPCM, Los resultados obtenidos se muestran en la tabla 6.10. La etapa de codificación Huffman no se evaluó, porque el comportamiento del algoritmo es muy variable.

Etapa de la Compresión	Tiempo de ejecución [µs]
Obtención de MCU y desplazamiento de nivel	12.110
DCT	147.155
Cuantización	17.168
Zigzag	2.561

Tabla 6.10 Tiempos de ejecución para las etapas de compresión sobre una unidad de datos

De acuerdo con los resultados obtenidos en la tabla 6.10, la DCT es la parte de la compresión que requiere mayor tiempo de procesamiento, por lo que en un proyecto a futuro podría optimizarse al ser programada en lenguaje ensamblador.

El proceso de obtención de las MCUs puede ser optimizado si se utilizan canales EDMA para hacer el ordenamiento sin que intervenga el CPU, y los procesos de desplazamiento de nivel, cuantización y zig-zag pueden ser optimizados programándose en lenguaje ensamblador y/o utilizar un buffer en la memoria interna del DSP, en vez de procesar los datos en la memoria externa, aunque todas estas optimizaciones aumentan de forma muy significativa la complejidad del sistema, así como su sincronización.

6.3.- Evaluación del Pipeline Hardware-Software del sistema

El sistema de adquisición y compresión de video MJPEG funciona como un *pipeline* hardware-software, como se muestra en la figura 6.13. El *pipeline* es una técnica de procesamiento en paralelo muy utilizada en los DSPs y consiste en traslapar operaciones durante el proceso de ejecución del sistema. En una arquitectura digital que opere con pipeline, las instrucciones son transferidas a estados sucesivos, donde unidades separadas de hardware o software realizan diversas operaciones para completar la ejecución de la instrucción [30].

Nuestro sistema consta de tres procesos principales o niveles: *adquisición de imagen, transferencia por EDMA* y *procesamiento JPEG*.

- La *adquisición de imagen* es el proceso en el cual el sensor de video captura una imagen, la envía a los buffers tipo FIFO y genera un pulso de sincronía vertical (VSync) cuando se ha capturado una imagen completa, generando una interrupción externa al DSP (*sección 4.4*).

- La *transferencia EDMA* comprende la transferencia de datos de la imagen desde los buffers tipo FIFO de la DSKCam al triple buffer de la memoria externa SDRAM del DSP (*secciones 4.5.2 y 4.5.3*).

- En el *proceso JPEG*, se incluye la adecuación de la imagen al formato QCIF con los datos de los componentes ordenados de forma separada y el proceso de compresión JPEG de una imagen completa.

El pipeline hardware-software del sistema funciona de la siguiente manera:

1) Al inicio, en el tiempo T_{I1} se adquiere la imagen I1, y no existe transferencia por EDMA ni imagen para procesar en JPEG.

2) Una vez que se ha adquirido una imagen completa, el sensor de video genera un pulso de sincronía vertical (*VSync*), el cual ocasiona una interrupción externa por hardware al DSP, para iniciar la transferencia EDMA en el tiempo T_{DMA1}. Mientras tanto, el sensor de video adquiere la imagen I2 en el tiempo T_{I2}. Es importante resaltar que los procesos de adquisición de imagen y de transferencia EDMA al inicio del tiempo T_{I2}, se ejecutan de forma simultánea.

3) Una vez que se tiene una imagen en el triple buffer de la memoria externa del DSP, el DSP aplica el procesamiento JPEG a la imagen I1, en el tiempo T_{P1}.

4) Durante parte del tiempo T_{P1}, comienza la adquisición de la imagen I3, y la transferencia por EDMA de la imagen I2. Cuando el DSP termina de procesar la imagen I1, empieza a procesar la imagen I2 en el tiempo T_{P2}.

5) Los procesos descritos anteriormente se repiten continuamente, formando el pipeline hardware-software del sistema.

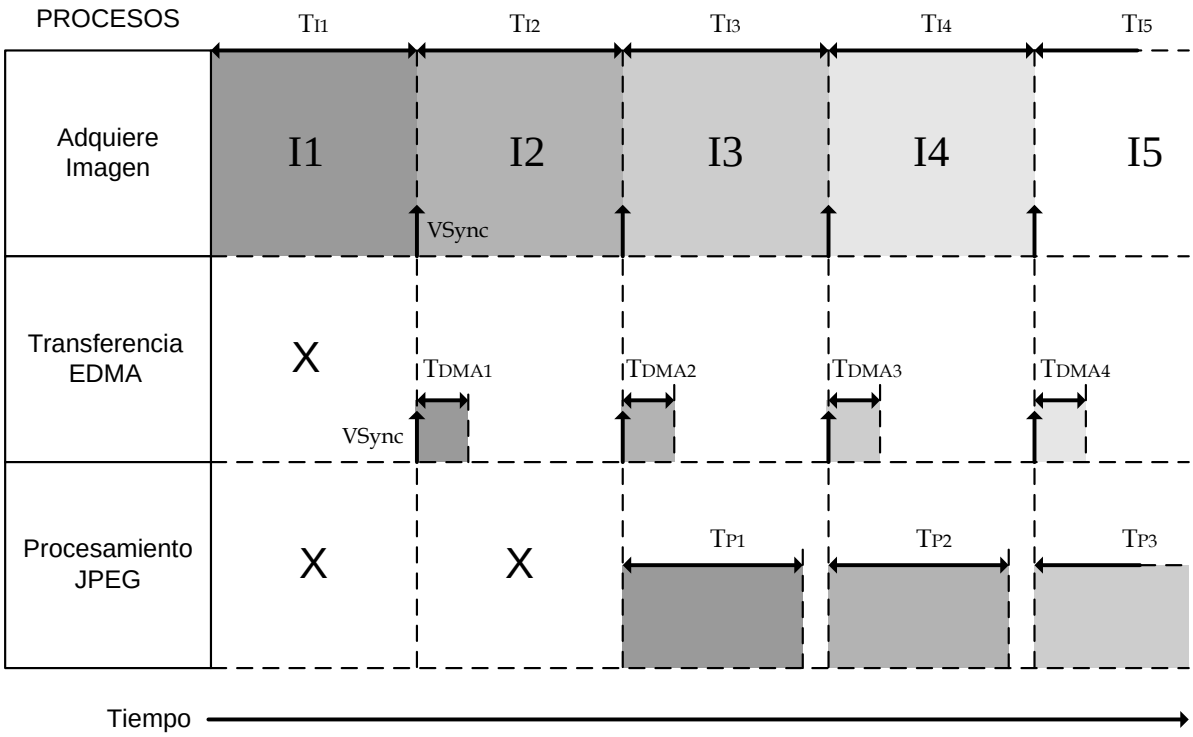


Figura 6.13 Pipeline hardware-software del sistema de adquisición y compresión de video MJPEG.

Consideraciones del pipeline hardware-software

El pipeline hardware-software mostrado en la figura 6.13 tiene la restricción de que solo funciona en tiempo real para video a color con formato QCIF 4:2:0 a una tasa de 6 cuadros por segundo, debido a que el tiempo entre cada cuadro es de 167 ms aproximadamente, y el tiempo de procesamiento JPEG es de 150 ms, 110 ms de procesamiento JPEG y 40 ms de la adecuación de la imagen (*tablas 6.1 y 6.7*). Cabe recordar que para estar dentro del tiempo real, el tiempo de procesamiento de la señal debe de ser menor al tiempo de muestreo de la señal, en nuestro caso consideramos cada imagen como muestra, y el tiempo entre la adquisición de cada imagen como el tiempo de muestreo.

Si en vez de video a color se utiliza video en blanco y negro, el tiempo de procesamiento JPEG sería de 82.8 ms, 74 ms de procesamiento JPEG y 8.8 ms de adecuación de la imagen (*tablas 6.1 y 6.9*), por lo que la tasa de cuadros por segundo que se puede manejar para estar dentro del tiempo real sería de 12 cuadros por segundo.

La tasa mínima de cuadros por segundo es de 10 fps para comunicaciones de video, o sistemas de vigilancia (*sección 1.5.2*), por lo que en el caso de adquisición y compresión de video MJPEG utilizando video blanco y negro, el sistema puede funcionar muy bien. En el caso del video a color se deben optimizar los tiempos de procesamiento, para poder mejorar la tasa de cuadros por segundo que puede manejar el sistema. Además, a futuro se pretende realizar la transmisión y la descompresión de video comprimido en tiempo real por algún otro dispositivo, como una PC.

Por otro lado, si al pipeline de la figura 6.13 le agregamos la transmisión serial propuesta en la *sección 5.3*, tendríamos un pipeline de cuatro niveles como se muestra en la figura 6.14, el cual funcionaría de manera similar al de la figura 6.13, solo que ahora tendría el nivel de transmisión serial, con un tiempo de transmisión T_T, el cual sería de 111 ms aproximadamente para un cuadro comprimido en blanco y negro con QF=0.5 y cerca de 155 ms para un cuadro comprimido a color con QF=0.5, tomando como referencia que la tasa máxima de transmisión permitida en los PCs es de 115,200 bps (*sección 5.3*), por lo que si se quiere operar en el tiempo real la tasa de cuadros por segundo tendría que reducirse un poco mas que en el caso del pipeline de tres niveles.

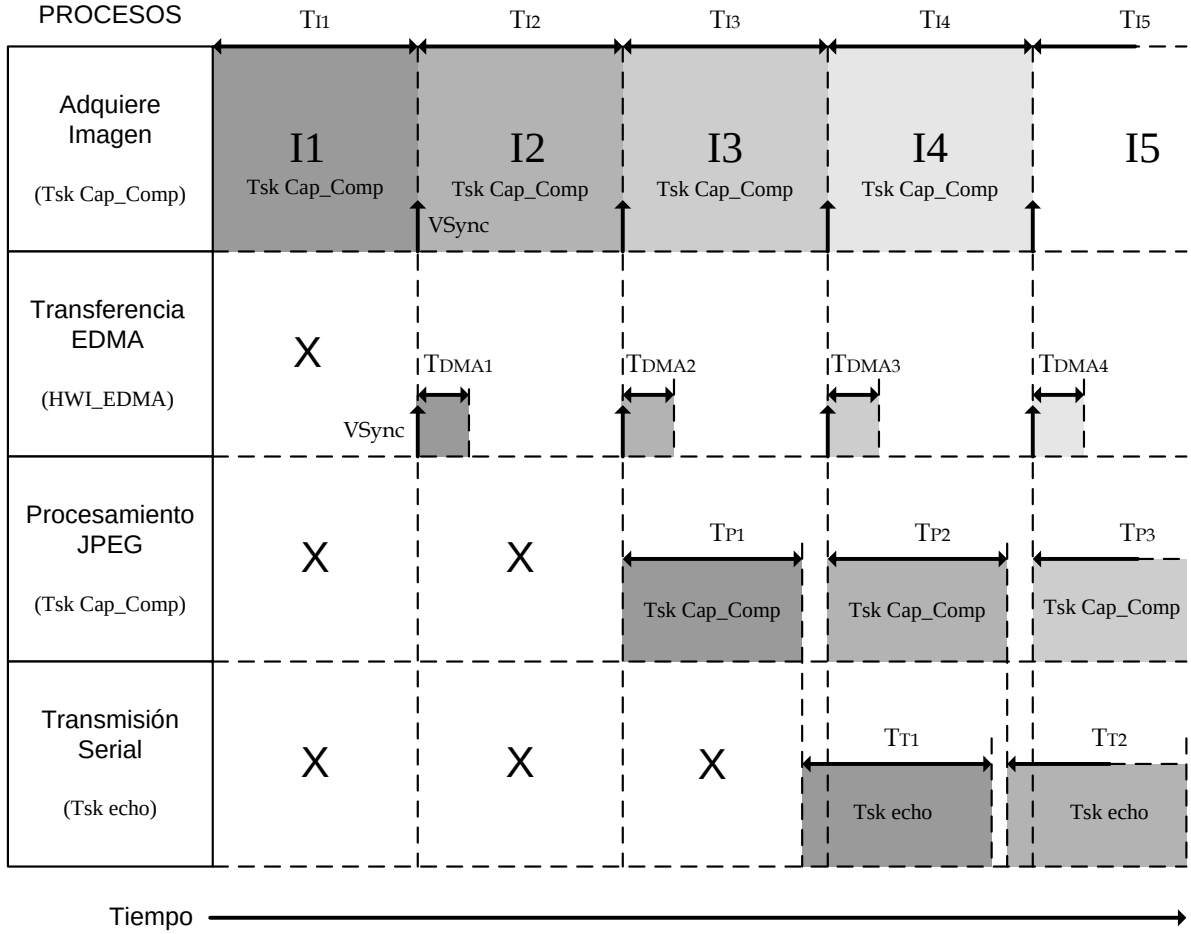


Figura 6.14 Pipeline hardware-software del sistema de adquisición, compresión y transmisión de video MJPEG.

RESUMEN

En este capítulo se presentó el análisis de resultados de la adquisición de video, los tiempos de transferencia por EDMA, tiempos de adecuación de la imagen y el espacio utilizado en memoria para distintos formatos de video. Además, se presentaron los resultados gráficos para diferentes secuencias de video adquiridas. En este capítulo, también se presentaron los resultados de la compresión de video MJPEG, siendo los principales los factores de compresión, el porcentaje de error y los tiempos de procesamiento. Finalmente, se hizo un análisis del pipeline hardware-software del sistema, evaluando su factibilidad y sus limitaciones para funcionar en tiempo real.

CONCLUSIONES

En este trabajo, se diseñó e implementó un sistema para adquisición y compresión de video utilizando el algoritmo MJPEG, haciendo uso de una cámara de video digital tipo CMOS y de la tarjeta de desarrollo del DSP C6416 (DSK6416).

Se logró adquirir imágenes de video a color en formato VGA 4:2:2 y QVGA 4:2:2, y realizar las adecuaciones necesarias para obtener secuencias de video con formato QCIF 4:2:2 y QCIF 4:2:0, así como secuencias de video en blanco y negro con formato QCIF. Con excepción del formato VGA, los tiempos de procesamiento y de transferencia por EDMA están dentro del tiempo real considerando una tasa de 30 fps, la cual es el estándar para televisión comercial, con muy buena calidad.

En cuanto a los resultados obtenidos en la compresión de video, al utilizar un factor de cuantización QF=0.5, se obtuvo una calidad de video recuperado regular, y un factor de compresión muy bajo, de aproximadamente 5% con respecto al tamaño del video original. El porcentaje de error cercano al 3% también es muy bueno.

Los tiempos de compresión obtenidos para video a color no fueron lo suficientemente óptimos, por lo que si se quisieran utilizar en tiempo real, la máxima tasa de cuadros por segundo que podría soportar el sistema es de 6 fps. En el caso de video a blanco y negro, la tasa máxima es de 12 fps para poder estar dentro del tiempo real.

El sistema de adquisición y compresión de video MJPEG desarrollado funciona como un pipeline hardware-software de 3 niveles, adquisición de imagen, transferencia por EDMA y compresión JPEG. Como se mencionó anteriormente, las limitaciones que tiene el sistema es que solo puede operar en tiempo real con video a color a 6 fps y con video en blanco y negro a 12 fps.

Con los resultados obtenidos en este proyecto para compresión de video en blanco y negro, si se agrega la etapa de transmisión, el sistema se podría utilizar en sistemas de vigilancia o para videoconferencias por Internet, ya que la tasa mínima de cuadros requerida para este tipo de aplicaciones es de 10 fps, con un formato de video QCIF.

Para mejorar el desempeño del sistema se tienen tres alternativas. La primera es optimizar los algoritmos para que sean más rápidos; la segunda es explotar toda la potencialidad del hardware utilizando canales EDMA para mover bloques de memoria y obtener las unidades mínimas de codificación (bloques de 8x8) sin la intervención del CPU. Otra solución es utilizar los nuevos DSPs con 25,000 MIPS, con el cual se puede mejorar hasta tres veces el desempeño del sistema, ya que tan solo sin realizar ninguna optimización algorítmica, el sistema actual soportaría una tasa de 18 fps con estos DSPs. La tercera opción para optimizar el sistema es programar la DCT en lenguaje ensamblador, ya que es la función de la compresión MJPEG que requiere más procesamiento y tratar de procesar las imágenes dentro de la memoria interna del DSP en vez de la memoria externa del DSP.

La finalidad del sistema a futuro es que se pueda adquirir, comprimir, transmitir, y en otro módulo similar descomprimir y visualizar video a color en tiempo real, de tal forma que con el sensor de video y el DSP se adquiera y comprima video de forma independiente, y se transmita el video por una red TCP/IP, y en el otro lado del sistema, una PC reciba el video comprimido, lo descomprima y lo despliegue en el monitor, todo esto en tiempo real, y tratando de que el tamaño de video sea lo más grande posible, siendo el tamaño QVGA una buena opción.

ANEXO A

Tablas de Codificación Huffman para la compresión

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.1 Códigos para la codificación de coeficientes DC de 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.2 Códigos base para la codificación de coeficientes DC de Crominancia

Cadena Categoría	Longitud del código	Código Huffman
0/0 (EOB)	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	11111110110
1/6	16	1111111110000100
1/7	16	1111111110000101
1/8	16	1111111110000110
1/9	16	1111111110000111
1/A	16	1111111110001000

Cadena Categoría	Longitud del código	Código Huffman
4/1	6	111011
4/2	10	1111111000
4/3	16	1111111110010110
4/4	16	1111111110010111
4/5	16	1111111110011000
4/6	16	1111111110011001
4/7	16	1111111110011010
4/8	16	1111111110011011
4/9	16	1111111110011100
4/A	16	1111111110011101
5/1	7	1111010
5/2	11	11111110111
5/3	16	1111111110011110
5/4	16	1111111110011111
5/5	16	1111111110100000
5/6	16	1111111110100001
5/7	16	1111111110100010
5/8	16	1111111110100011
5/9	16	1111111110100100
5/A	16	1111111110100101

Cadena Categoría	Longitud del código	Código Huffman	Cadena Categoría	Longitud del código	Código Huffman
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
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

Cadena Categoría	Longitud del código	Código Huffman	Cadena Categoría	Longitud del código	Código Huffman
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.3 Códigos para la codificación de coeficientes AC de la 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***
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.4 Códigos para la codificación de coeficientes AC de la Crominancia

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 A.5 Tabla de 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 A.6 Tabla de cuantización de Crominancia

ANEXO B

Programa principal para la sincronización de tareas del sistema

```
/*************************************************************************
*************************************************************************
*
*                              Empieza función   main
*
*************************************************************************
*************************************************************************/

main(){

/* Declara parámetros necesarios para la cámara DSKcam*/
 DSKcam_params CAM_params =  {

#if CHIP_6711
  0,    /* segid para DSK6711 */
#else
  1,    /* segid DSK6416 & DSK6713 */
#endif

  QVGA, /* Tamaño de Cuadro */
  NULL  /* Registros de usuarios de la cámara */
  };

/* Inicializa la cámara */
 if( DSKcam_open(&CAM_params) ) { SYS_abort("DSKcam_CAMinit"); }

 /*Función global que habilita interrupciones */
 IRQ_globalEnable();

}    // Termina main


/************************************************************************
*     Función para la captura de video                                  *
************************************************************************/
void captura()
{
 /* Declara apuntador tipo char necesario para la Fuente de Cuadros */
 unsigned char * FuenteCuadro;

 while(1==1)
 {
   LOG_printf(&trace, "Task captura ejecuta SEM_pend");
   SEM_pend(&sem_prin, SYS_FOREVER);

/*  Rutina para almacenar un número de cuadros = totalCuad en memoria SDRAM
externa*/
   if (numCuadro<totalCuad)
      {
       FuenteCuadro = Extrae_cuadro();
```

```
       orden_QCIF_420(DSKcam_QVGA_WIDTH * DSKcam_QVGA_HEIGHT, SrcFrame,
      Cuadro[numCuadro].Y, Cuadro[numCuadro].Cr, Cuadro[numCuadro].Cb);
       LOG_printf(&trace, "Cuadro capturado y ordenado");
       numCuadro += 1;   /* Incrementa numero de cuadro */
       }
   else
     {
     //****Inicia compresor*****
       compresion (Cuadro[0].Y, Cuadro[0].Cr, Cuadro[0].Cb);
       numCuadro = 0; //Reinicia numCuadro
      }   // Termina compresor (fin de If para capturar o comprimir)
  }    // Termina while (1==1)
} //Termina funcion captura

/*****************************************************************************
                  Función para la transmisión de imágenes por puerto serie
*****************************************************************************/
Void echo()
{
      Uns         size = 38016;
      GIO_Handle  inChan, outChan;
      pbuf = buf;
      vect = Cuadro[0].Y;
      countbuff=0;
      inChan = GIO_create("/uart", IOM_INPUT, NULL, NULL, NULL);
      outChan = GIO_create("/uart", IOM_OUTPUT, NULL, NULL, NULL);

if (inChan == NULL || outChan == NULL) {
        LOG_printf(&trace, "GIO_create failed");
        SYS_abort("GIO_create");
    }
while (1==1)
   {
         LOG_printf(&trace, "Task echo ejecuta SEM_post");
         SEM_post(&sem_prin);
         status = GIO_write(outChan, pvect, &size);
         LOG_printf(&trace, "Inicia transmision de cuadro");
         if (status < 0)
          {
                LOG_printf(&trace, "GIO_write error");
                SYS_abort("GIO_write");
          }
    /*Rutina para buffer circular de 10 cuadros */
        LOG_printf(&trace, "Termina transmision de cuadro");
       if (countbuff < 20)
        {
          pvect = pvect + 38016;
          countbuff = countbuff +1;
        }
          else
           {
            pvect = Cuadro[0].Y;
               countbuff = 0;
             }
      } //Fin del While (1==1)
} // Fin de echo
```

BIBLIOGRAFIA

[1] ***"Video Codec Design",*** Ian E. G. Richardson, Wiley, E.U., 2002

[2] ***"Image processing with LabVIEW and IMAQ Vision",*** Thomas Klinger, Prentice Hall, E.U., 2003

[3] ***"Compressed Image File Formats JPEG, PNG, GIF, XBM, BMP",*** John Miano, Addison Wesley, New York, E.U., 2005

[4] ***"Sistemas Audiovisuales: 1.- Televisión Analógica y Digital",*** Francesc Tarrés Ruiz, Ediciones UPC, Barcelona, España, 2000

[5] ***"Video Demystified",*** Keith Jack, Newnes, 4a. Ed., E.U., 2005

[6] ***"The Technology of Video & Audio streaming"***, David Austerberry, Focal Press, 2a. Ed., E.U., 2005

[7] ***"CCD vs CMOS: Facts and Fiction",*** David Litwiller, Photonics Spectra, Laurin Publishing, Canada, 2001

[8] ***"Dictionary of Video and Television"***, Newnes, E.U., 2002

[9] ***"Microprocessors and Interfacing: Programming and Hardware",*** Douglas V. Hall, Mc Graw Hill, Singapore, 1986

[10] ***"TMS320C6414T, TMS320C6415T, TMS320C6416T, Fixed-Point Digital Signal Processors" (SPRS226J)*** Texas Instruments, 2003

[11] ***"Standard Codecs: Image Compression to Advanced Video Coding",*** Mohammed Ghanbari, Institution of Electrical Engineers, Reino Unido, 2003

[12] ***"H.264 and MPEG-4 Video Compression"***, Iain E. G. Richardson, Wiley, Inglaterra, 2003

[13] ***"Image and Video Compression for Multimedia Engineering"***, Yun Q. Shi, Huifang Sun, CRC, E.U., 2000

[14] ***"Data Compression"***, David Salomón, Springer, 3a. Ed., New York, E.U., 2004

[15] ***"Handbook of Image and Video Processing"***, Jerry D. Gibson, et. al., Academic Press, Canada, 2000

[16] ***"Compresión MJPEG de video digital en un DSP",*** Marcos de Jesús Vitela Rodríguez, Tesis, México D.F., 2007

[17] ***"Real-Time Video Compression"***, Raymond Westwater,
Kluwer Academic Publishers, E.U., 1997

[18] ***"Information technology – Digital compression and coding of continuous-tone still images – Requirements and guidelines",*** CCITT Recommendation T.81, ITU, 1993

[19] ***"JPEG Still Image Data Compression Standard"***, William B. Pennebaker, Joan L. Mitchell, Chapman & Hall, New York, 1993

[20] ***"Discrete Cosine Transform, Algorithms, Advantages, Applications"***, K. R. Rao, P. Yip, Academic Press, E.U., 1990

[21] ***"TMS320C6416 DSK Technical Reference"***, Spectrum Digital, E.U., 2004

[22] ***"DSKcam Users manual Version 3.1"***, BiTec Ltd, Inglaterra, 2005

[23] ***"TMS320C6000 DSP Enhanced Direct Memory Access (EDMA) Controller Reference Guide" (SPRU234)***, Texas Instruments, 2006

[24] ***"Serial Camera Control Bus Functional Specification"***, OmniVision Technologies, 2002

[25] ***"OV7620 single-chip CMOS VGA color digital camera"***, OmniVision Technologies, 2000

[26] ***"Real-Time Digital Signal Processing"***, Nasser Kehtarnavaz, Newnes, E.U., 2005

[27] ***"Digital Signal Processing and Applications with the C6713 and C6416 DSK"***,
R. Chassaing, Wiley,E.U., 2005

[28] ***"Serial Port Complete - Programming and Circuits for RS-232 and RS-485 Links and Networks"***, Jan Axelson, Lakeview Research, Madison, E.U., 2000

[29] ***"TMS320C6000 DSP/BIOS User's Guide" (SPRU303B)***, Texas Instruments, 2000

[30] ***"Arquitectura de DSPs familias TMS320C54x y TMS320C54XX y aplicaciones"***, Larry Escobar Salguero, Bohumil Psenicka, Miguel Molero Armenta, Facultad de Ingeniería, U.N.A.M., 2005

[31] ***"Codificador Decodificador MJPEG"***, Larry Escobar Salguero, Edgar Alonso Trueba y Marcos Vitela Rodríguez, XII Congreso de Instrumentación SOMIXXII, México, 2007

[32] ***"Implementación de un Codec MJPEG con aproximación al Tiempo Real"***, Larry Escobar Salguero, Edgar Alonso Trueba y Marcos Vitela Rodríguez, 3er Congreso Nacional de Ingeniería Mecánica, Eléctrica, Electrónica, y Mecatrónica (CIMEEM 2008), UAM, México, 2008