

## Implementación hardware de algoritmos de procesado de imagen



Autor: Ramón Blanco Caamaño

Tutora: María Dolores Valdés Peña

Curso: 2013-2014

## Contenido.

1. Introducción.....	3
2. Objetivos.....	3
3. Procedimiento de trabajo.....	4
4. Resultados.....	5
4.1 Estado del arte.....	5
4.2 Captura de imágenes.....	5
4.3 Marco teórico.....	6
4.4 Prototipo en Matlab.....	6
4.5 Implementación del proyecto.....	7
4.5.1 Captura de la imagen.....	8
4.5.2 Algoritmo para la conversión de imágenes RGB a HSL.....	9
4.5.3 Algoritmo para la creación de un histograma HSL.....	9
4.5.4 Almacenamiento del histograma en una memoria RAM.....	10
4.5.5 Cálculo de distancias entre histogramas.....	10
5. Conclusiones.....	11
6. Bibliografía.....	12
Anexo 1.....	15
Anexo 2.....	23
Anexo 3.....	38
Anexo 4.....	65
Anexo 5.....	95

## 1. Introducción.

Cada vez más somos una sociedad dependiente del avance tecnológico que va desplazando los roles clásicos de los seres humanos. La digitalización de la imagen y su estrecha relación con la electrónica y el cómputo con ordenadores, ha generado nuevas aplicaciones y productos innovadores. En algunos casos ya no es necesario un operario que procese una situación o escenografía, pues con la aparición de la robótica y de la inteligencia artificial, una máquina puede tomar numerosas decisiones a través de una amplia gama de sensores e imágenes digitales.

Los equipos electrónicos son cada vez más sofisticados y complejos, su manejo es a más “alto nivel” (interfaces más amigables). Estos equipos, con la ayuda del procesamiento digital de imágenes, podrán identificar figuras u objetos y realizar desde tareas básicas hasta complejas. Incluso desafían la eficacia y los límites humanos, una máquina es extremadamente fiable y posee características útiles que no pueden despeñar los humanos. Por ejemplo: visión infrarroja, identificación de figuras concretas de manera muy precisa, detección de impurezas microscópicas, conteo de objetos de manera instantánea, colorimetría objetiva, etc.

Así surge la visión artificial, también conocida como visión por cómputo o visión técnica, una combinación de la fotografía y el video digital con la inteligencia artificial (figura 1).



*Figura 1: Ejemplo de visión artificial.*

## 2. Objetivos.

El objetivo principal de este Trabajo Fin de Grado (TFG) es el desarrollo de una solución hardware basada en FPGA para la implementación de algoritmos de procesado digital de imagen. Esto incluye el estudio de los distintos métodos “clásicos” utilizados en el procesado digital de imagen, y posteriormente, la propuesta de una aplicación concreta y su implementación hardware mediante FPGA. Como aplicación práctica se optó por la implementación de un sistema capaz de determinar el grado de similitud de dos imágenes, utilizando para ello técnicas de visión artificial.

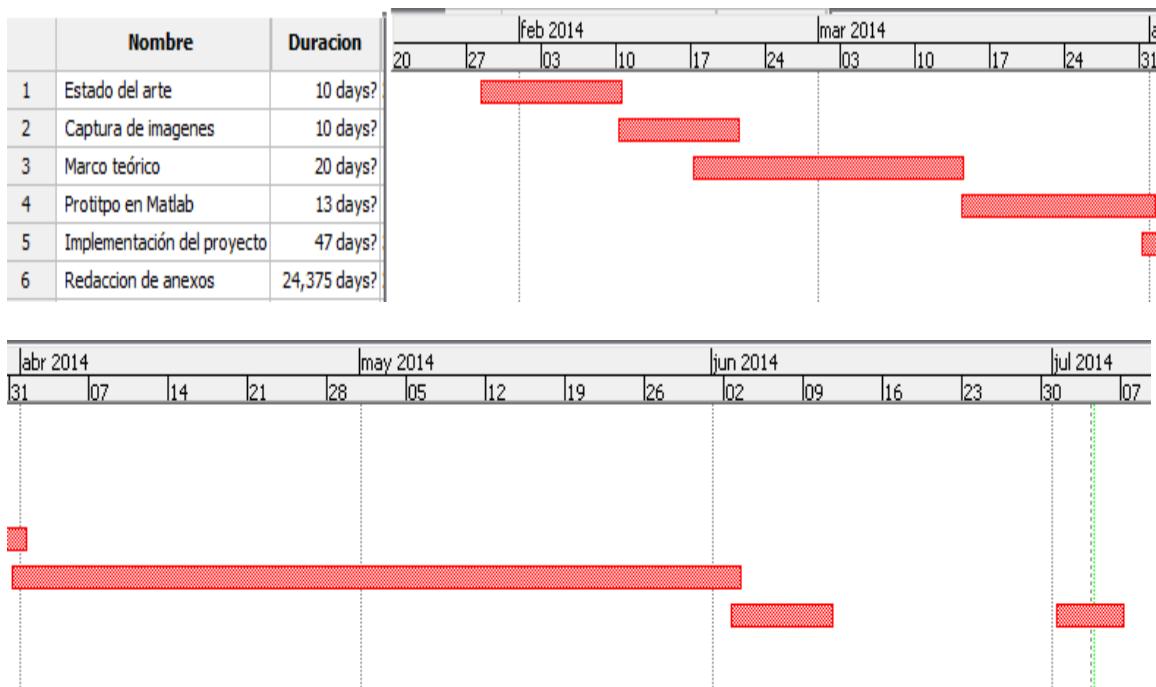
Para llevar a cabo este objetivo general se definieron los siguientes objetivos parciales:

1. Estudio de los conceptos relacionados con la visión artificial y definición del estado del arte de esta tecnología.
2. Estudio de los dispositivos de captura de imagen. Estudio del módulo *VmodCAM* del fabricante *Digilent*.
3. Estudio de los algoritmos básicos de procesado de imagen y definición de una aplicación práctica que utilice dichos algoritmos.
4. Diseño teórico del sistema. Incluye la definición de la arquitectura básica del sistema a diseñar y los bloques funcionales que lo componen.
5. Prototipado del sistema mediante *Matlab*.
6. Diseño del sistema hardware mediante el lenguaje VHDL y las herramientas de desarrollo de *Xilinx* para FPGA. Verificación del sistema diseñado.

Para la realización del proyecto se parte de los conocimientos adquiridos en la rama de sistemas electrónicos del Grado en Ingeniería de Tecnologías de Telecomunicación.

### 3. Procedimiento de trabajo.

El procedimiento de trabajo empleado para la realización del TFG consistió en la definición de un conjunto de tareas conducentes a resultados parciales. En la figura 2 se indican las tareas concretas realizadas y su distribución temporal.



*Figura 2: Temporización de las tareas realizadas en el TFG.*

## 4. Resultados.

Teniendo en cuenta los objetivos del trabajo y el procedimiento de elaboración utilizado, a continuación se resumen los resultados parciales y el resultado final del trabajo realizado. Los resultados se organizan siguiendo el orden cronológico de las tareas ejecutadas.

### 4.1 Estado del arte.

Se investigaron desarrollos de última tecnología realizados sobre el procesado digital, que han sido probados en la industria y acogidos y aceptados por diferentes fabricantes.

Así mismo, se analizaron las características de los dispositivos configurables (FPGAs) poniendo especial énfasis en el estudio de la familia Spartan-6 de Xilinx, que es la que se ha utilizado en este trabajo. Se evaluaron los recursos hardware disponibles de cara a poder utilizarlos de forma eficiente en la fase de implementación del sistema.

En el *Anexo 1. Estado del arte* se hace un análisis sobre la viabilidad de soluciones software y hardware en aplicaciones de procesado de señal, y se resumen algunas de las características más importantes de las FPGAs actuales que las hacen idóneas para aplicaciones de este tipo.

En esta fase se llegó a las siguientes conclusiones:

- Cuando los requisitos de funcionamiento del sistema exceden las capacidades de los procesadores DSP existentes, el desarrollo de un hardware específico es la única alternativa viable.
- Las FPGAs son una alternativa válida para el soporte hardware de aplicaciones de procesado de señal.

### 4.2 Captura de imágenes.

Se estudiaron las diferentes cámaras digitales comerciales utilizadas para captar imágenes y videos de una escena. Lo cual permitió profundizar en los conocimientos adquiridos en la asignatura de segundo curso *Fundamentos de Sonido e Imagen* sobre el funcionamiento de las cámaras. Como resultado aprendimos a seleccionar la cámara idónea para cada tipo de escena que queremos captar. En el *Anexo 2. Captura de imágenes*, se incluye la información relevante sobre esta temática.

Uno de los objetivos del proyecto es conocer y llegar a manejar el módulo VmodCAM de *Digilent*. En este sentido, se realizó el estudio de las hojas de característica del módulo y del sensor de imagen disponible en dicho modulo. Las características más importantes se resumen en el *Anexo 2. Captura de imágenes*.

Con el objetivo de controlar el módulo VmodCAM, es decir, controlar la captura de una imagen y su envío a un monitor para su visualización, se tomó como referencia

el sistema VHDL propuesto por *Digilent*. A dicho sistema se añadió un bloque de control y una memoria RAM que permitió la captura controlada de una imagen.

## 4.3 Marco teórico.

Estudio de las técnicas más comunes del procesado digital de imagen. Asentamos las bases teóricas necesarias para poder enfrentarnos al proyecto, por ello, indagamos sobre los diferentes conceptos teóricos que envuelven a la representación y el procesado de la imagen digital.

- Se estudió cómo se obtienen y cómo se representan las imágenes digitales en los diferentes sistemas visuales e informáticos actuales.
- Se analizaron las ventajas e inconvenientes de las diferentes colorimetrías utilizadas en el procesado de imagen.
- Se estudió la utilización de histogramas para obtener una visión general de la distribución de píxeles que conforman una imagen.
- Se realizó un estudio teórico de filtros basado en la primera y segunda derivada: *Sobel*, *Prewitt* y *Roberts*, o *Canny* en el caso de los filtros basados en la segunda derivada.
- Se investigó cómo calcular la distancia (grado de similitud) entre dos distribuciones de valores diferentes: intersección, correlación, Chi-cuadrado y Bhattacharyya.

El Anexo 3: *Marco teórico*, recoge toda la información necesaria para la iniciación al procesado digital de imágenes. El anexo concluye con la definición y la justificación de una aplicación práctica que utilice dichos algoritmos.

## 4.4 Prototipo en Matlab.

Durante esta tarea del proyecto hemos realizado un pequeño estudio teórico-práctico para la implementación de un prototipo del algoritmo de procesado. Para conseguir dicho fin, se hizo uso de *Matlab*, una excelente herramienta de software matemático que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio. *Matlab* posee una caja de herramienta específica para el procesado de imagen.

La primera fase de esta tarea fue el aprendizaje de las nociones básicas del *Toolbox* de procesado de imagen en *Matlab*: lectura y escritura de imágenes, tipos de datos y de imágenes, modificación de las imágenes, representación de gráficas e histogramas, etc.

Una vez adquiridos los conocimientos necesarios, iniciamos una serie de pruebas y ensayos con el fin de entender cada uno de los algoritmos que se deben implementar en la aplicación. En la documentación desarrollada durante esta fase de prototipado se comprueba cada una de las alternativas consideradas viables en el marco teórico y se verifica su correcto funcionamiento. Por lo tanto, aparte de definir un algoritmo definitivo para cada una de las partes que constituyen la aplicación, en esta fase se simula y se comprueba la viabilidad de los algoritmos desde el punto de

vista de su implementación práctica. De esta forma podemos evaluar de manera aproximada el resultado esperado en la aplicación y valorar las ventajas e inconvenientes del modelo elegido.

En el *Anexo 4. Prototipo en Matlab*, se incluye toda la información necesaria para el desarrollo de los algoritmos implementados en la aplicación. Se realiza una simulación y prototipado del resultado esperado en la solución hardware basada en FPGA.

## 4.5 Implementación del proyecto.

El experimento transcurre en una escenografía concreta, dónde el usuario va colocando y comparando diferentes objetos monitorizados por una cámara. El usuario toma una fotografía de referencia de un objeto concreto y posteriormente toma diversas imágenes, que serán procesadas y comparadas con la imagen de referencia almacenada. Por lo tanto, el resultado esperado es un sistema hardware que debe ser capaz de almacenar en la memoria una escena de referencia y calcular el grado de similitud con las siguientes escenas fotografiadas.

El material disponible para el desarrollo del proyecto es: un módulo *VmodCAM* de *Digilent*, una placa de desarrollo Atlys de *Digilent* que incluye una FPGA de la familia Spartan-6 de *Xilinx* y un monitor con entrada de vídeo HDMI (figura 3).

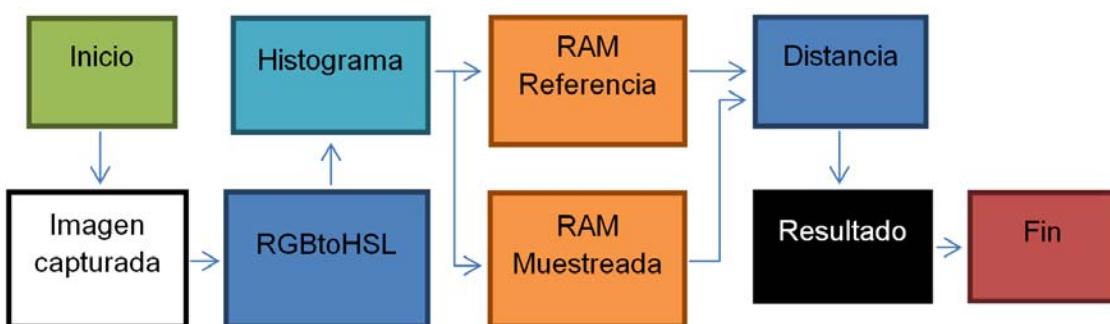


*Figura 3: Módulo VmodCAM conectado a la FPGA mediante el conector VHDCI.*

- **Módulo VmodCAM:** Es una cámara CMOS de 2 megapíxeles, con la que podemos obtener imágenes con resolución de hasta 1600x1200 píxeles y una velocidad de muestreo de 15FPS, dependiendo de la configuración utilizada. Está conectada a la FPGA mediante un conector VHDCI.

- **FPGA Spartan-6:** La *FPGA* de la familia Spartan ® -6 de Xilinx ofrece una gran escala de integración que nos permite el desarrollo de aplicaciones de alta exigencia a un coste reducido. Para más información revisar la hoja de característica de la *FPGA* utilizada, así como los Anexos 1 y 5 (*Estado del arte e Implementación hardware del proyecto*).
- **Monitor:** El monitor está conectado a la *FPGA* mediante HDMI, a través de él podemos visualizar las imágenes capturadas por la cámara.

En los siguientes párrafos resumo y explico paso a paso la arquitectura del sistema diseñado en la *FPGA*. En la figura 4 se muestra un diagrama de flujo de los bloques funcionales que lo componen.



**Figura 4. Bloques funcionales necesarios la implementación de la aplicación desarrollada.**

No profundizaré en detalles muy teóricos ni muy técnicos sobre la implementación de la aplicación, para entrar más a fondo en cada uno de los bloques que componen la aplicación, es necesaria la consulta de los anexos: Captura de imágenes, Marco teórico y Prototipo en *Matlab*.

Una vez finalizada la explicación, es esencial la lectura del Anexo 5. *Implementación hardware del proyecto*, así como ver el código VHDL implementado para la resolución definitiva del Trabajo Fin de Grado. Los anexos 3, 4 y 5 resumen todo los detalles técnicos relacionados con la generación del código VHDL resultante, así como las simulaciones empleadas para la comprobación de los bloques diseñados a lo largo de estos meses de trabajo.

#### 4.5.1      **Captura de la imagen.**

La cámara capture las imágenes o fotografías que vamos a utilizar durante el procesado. Está formada por una cámara oscura cerrada, con una abertura en uno de los extremos para que pueda entrar la luz y una superficie plana donde se formará la imagen (sensor CMOS). La cámara utilizada posee un objetivo formado por lentes, ubicado en la abertura para controlar la luz entrante y para enfocar la imagen, o parte de ella. Una vez obtenida la imagen es recibida en la *FPGA* a través del puerto VHDCI. Para más detalles revisar el Anexo 2. *Captura de imágenes*.

La fotografía recibida es una imagen en mapa de bits, que no es más que una matriz de píxeles. La matriz obtenida mediante la integración total de cada uno de

dichos píxeles representa la imagen. El modelo de color que utiliza la cámara empleada en el proyecto es el RGB565, un modelo de color aditivo con el que es posible la representación de cualquier color mediante la mezcla por adición de los tres colores primarios de la luz (Rojo + Verde + Azul). Para más detalles consultar *Anexo 3. Marco teórico*, apartado 2. “Tipos de imágenes digitales” y apartado 3.1 “Modelo RGB.”.

#### 4.5.2 Algoritmo para la conversión de imágenes RGB a HSL.

En este bloque transformamos el modelo de color RGB al modelo de color HSL. El modelo HSL extrae la componente lumínica de la imagen, desligándonos de dicha dependencia. Por lo tanto, podemos tener una escenografía mucho más complicada (peor iluminada) y conseguir la identificación de objetos de manera más eficiente (ver figura 5).

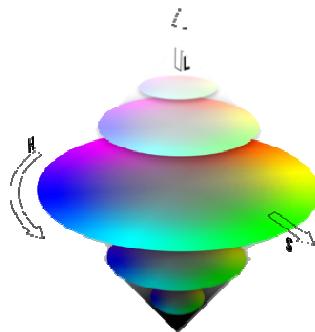


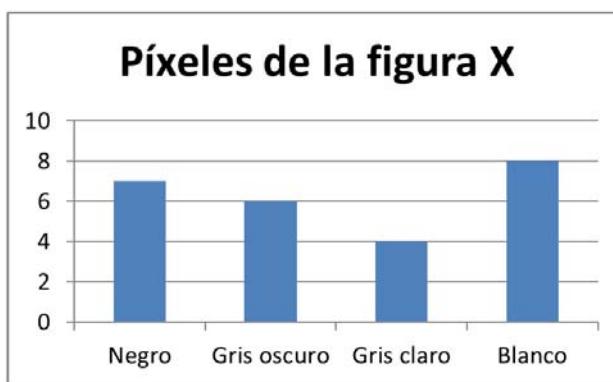
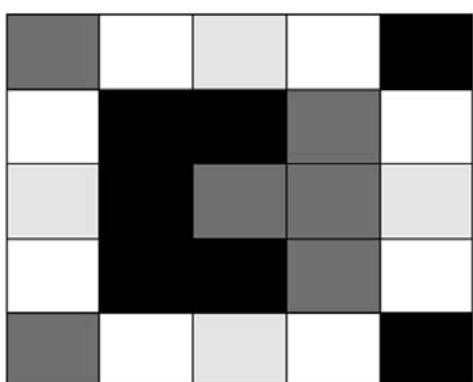
Figura 5. Modelo de color HSL

El modelo de color HSL es más cercano a la idea humana del color al añadir términos de tono, saturación y luminosidad. Se representa gráficamente como un cono doble dónde los dos vértices corresponden con el blanco y el negro respectivamente. El ángulo se corresponde con el tono, la distancia al eje con la saturación y la distancia al vértice blanco se corresponde con la luminancia.

Para más información consultar el *Anexo 3. Marco teórico* apartado 3.3 “modelo HSL” y apartado 3.5 “Conclusión sobre el modelo de color a utilizar en el proyecto”. El algoritmo desarrollado se estudia y verifica detenidamente en Matlab en el *Anexo 4. Prototipo mediante Matlab* apartado 2.7 “Algoritmo para la conversión de imágenes RGB a HSL”.

#### 4.5.3 Algoritmo para la creación de un histograma HSL.

El siguiente paso es la obtención del histograma para cada uno de los componentes que constituyen el modelo de color HSL. Un histograma es una representación gráfica de una variable en forma de barras, donde la superficie de cada barra corresponde a la frecuencia de los valores representados (ver figura 6).



**Figura 6. Histograma de una imagen.**

Son utilizados para obtener una visión general de la distribución de elementos que conforman la imagen, de esta manera, caracterizamos los comportamientos observados: el grado de homogeneidad, variabilidad, etc.

Para más información consultar el Anexo 3. *Marco teórico* apartado 4.1.1 “Histogramas” y apartado 4.2 “Conclusión sobre el modelo de procesado de imagen a utilizar en el proyecto”. El algoritmo desarrollado es estudiado y probado detenidamente en Matlab y está documentado en el Anexo 4. *Prototipo mediante Matlab* apartado 3.3 “Algoritmo para la creación de un histograma HSL”.

#### 4.5.4 Almacenamiento del histograma en una memoria RAM.

Una vez procesada la imagen se almacena en una memoria RAM interna de la FPGA llamada de “referencia”, para el caso de la imagen de referencia, o “muestreada”, para el resto de imágenes.

#### 4.5.5 Cálculo de distancias entre histogramas.

En este último apartado realizaremos la comparación de los histogramas, uno de referencia y otro muestreado. La finalidad es calcular el grado de similitud (distancia) entre los histogramas obtenidos. La métrica que hemos utilizado para determinar la similitud de los histogramas es la prueba  $\chi^2$  de Pearson, también llamada *Chi-cuadrado*.

Para más información consultar el Anexo 3. *Marco teórico* apartado 5. “Distancias entre la imagen de referencia y la imagen muestreada”, apartado 5.1.2 “Prueba  $\chi^2$ ” y apartado 5.2 “Conclusión sobre la métrica a utilizar para el cálculo de distancias”. El algoritmo desarrollado es estudiado y probado detenidamente en Matlab en el Anexo 4. *Prototipo mediante Matlab* apartado 3.3 “Algoritmo para la creación de un histograma HSL”.

## 5. Conclusiones.

Una vez finalizado el trabajo podemos concluir que el objetivo inicialmente planteado se alcanzó con éxito:

- Se analizaron las características del módulo VmodCAM y se desarrolló un módulo de control para la adquisición de imágenes.
- Se realizó un estudio de las técnicas de procesado de señal.
- Se propuso una aplicación concreta en la que se podían implementar en una FPGA algunas de las técnicas previamente analizadas.
- Se propuso una estructura de sistema y se desarrolló un modelo en Matlab, no sólo utilizando las librerías de procesado de señal disponibles en este lenguaje, sino desarrollando funciones propias con una estructura modular que pudieran ser reproducidas en un diseño hardware.
- Finalmente se desarrollaron los módulos hardware que constituyen el sistema en la FPGA y se realizó la puesta a punto del sistema mediante simulaciones temporales.

Teniendo en cuenta que el TFG se planteó desde un inicio como una toma de contacto con las tecnologías de procesado de imagen, la solución propuesta para determinar el grado de similitud de dos imágenes, es muy sencilla. No tiene en cuenta posibles funciones de pre-procesado, como por ejemplo, separar la imagen de interés de un fondo o suavizar la imagen para disminuir los efectos de iluminación. No obstante, desde el punto de vista didáctico constituye una aplicación muy válida.

Entre las posibles mejoras o líneas futuras de trabajo podríamos mencionar:

- Partiendo de la solución propuesta en este proyecto, basada en la comparación de histogramas, una tarea que se puede abordar de forma inmediata es la implementación de otras funciones de cálculo de distancias entre histogramas, como la intersección y el método de Bhattacharyya.
- Implementar otros algoritmos de procesado como por ejemplo los de detección de borde basados en la primera derivada de los niveles de gris de una imagen.
- Implementar funciones de pre-procesado para separar la imagen de interés del fondo.
- Dotar al sistema implementado de una interfaz de salida que permita mostrar los resultados del procesado de forma cómoda.

Desde el punto de vista de la aportación personal la realización de este proyecto ha supuesto un valor añadido en la formación como graduado en Ingeniería de Tecnologías de Telecomunicación. Ha supuesto una experiencia en la realización de un proyecto técnico que involucra diferentes tecnologías, algunas de ellas nuevas con respecto a la especialidad cursada.

## 6. Bibliografía.

- [1] Rodriguez-Andina, J.J., Moure, M.J., Valdes, M.D., “Features, Design Tools, and Application Domains of FPGAs”, IEEE Transactions on Industrial Electronics, vol. 54, no. 4, pp. 1810 – 1823, 2007.
- [2] Hojas de características del módulo VmodCAM,  
<http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,648,931&Prod=VMOD-CAM>
- [3] McClellan J.H., Schafer R.W., Yoder M.A., *Signal Processing First*, Ed. Pearson, 2003.
- [4] González R.C., Woods R.E., Eddins S.L., *Digital Image Processing Using Matlab*, Ed. Pearson, 2004.
- [5] González R.C., Woods R.E., *Digital Image Processing*, Ed. Pearson, 2002.
- [6] Cadenas J.O., Sherratt R.S., Huerta P., Kao W., “Parallel Pipelined Array Architectures for Real-time Histogram Computation in Consumer Devices”, IEEE Transactions on Consumer Electronics, vol. 57, no. 4, pp. 1460-1464, 2011.
- [7] Xu Q., Varadarajan S., Chakrabarti Ch. Karam L.J., “A Distributed Canny Edge Detector: Algorithm and FPGA Implementation”, IEEE Transaction on Image Processing, vol. 23, no. 7, pp. 2944-2960, 2014.
- [8] Asha V., Bhajantri N.U., Nagabhushan P., “GLCM based Chi-square Histogram Distance for Automatic Detection of Defects on Patterned Textures”, International Journal of Computational Vision and Robotics, vol. 2, no. 4, pp. 302-313, 2011.
- [9] Kailath T, “The Divergence and Bhattacharyya Distance Measures in Signal Selection”, IEEE Transactions on Communication Technology, vol. COM-15, no. 1, pp. 52-60, 1967.
- [10] Introducción á Toolbox de Imaxe de Matlab,  
[http://www.gts.tsc.uvigo.es/pi/practicas/intro\\_toolbox.pdf](http://www.gts.tsc.uvigo.es/pi/practicas/intro_toolbox.pdf)
- [11] Hojas de características de la familia Spartan-6,  
[http://www.xilinx.com/support/documentation/data\\_sheets/ds160.pdf](http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf)
- [12] Hojas de características del sistema de desarrollo Atlys,  
<http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,836&Prod=ATLYS>



## ANEXO 1: Estado del arte

### Contenido

1.	Introducción.....	15
2.	Hardware versus Software.....	16
3.	Dispositivos lógicos programables.....	17
4.	Field Programmable Gate Array “FPGA”.....	18
5.	Conclusión.....	20

## 1. Introducción.

Los primeros procesos industriales requerían de una gran mano de obra poco cualificada que realizaba tareas sencillas y repetitivas. Con el avance de la tecnología poco a poco se logró reemplazar dichos trabajadores por máquinas mucho más rápidas y eficientes. Uno de los procesos más importantes de la fabricación industrial es la inspección de los productos fabricados, para comprobar si cumplen o no las especificaciones previamente establecidas. Las tareas de verificación y validación siempre han sido una tarea prácticamente irreemplazable de los inspectores humanos.

Las habilidades para el patrón de reconocimiento humano, reconocimiento de lenguaje y producción de lenguaje se encuentran más allá de cualquier expectativa de los ingenieros de automatización. No obstante, actualmente ningún dispositivo puede competir con la precisión y certeza del ojo humano en muchas tareas. Es cierto que las cámaras digitales funcional igual o mejor que el ojo a la hora de captar imágenes pero la inteligencia artificial aún no ha podido igualar al cerebro humano.

Con la aparición de la visión artificial poco a poco se han conseguido pequeños avances de cara a la substitución de la inspección humana en los procesos industriales. Las ventajas de la visión por cómputo son sobresalientes en términos de aumento de homogeneidad de los resultados y la posibilidad de la inspección visual en situaciones extremas para la vista humana: inspección microscópica, infrarrojos, ausencia de iluminación, ambientes peligrosos, velocidad de trabajo constante, etc.

A pesar de las grandes ventajas de la visión por cómputo, la implantación en la producción industrial es muy baja. Las principales problemáticas para la implantación de dicha técnica son la complejidad, el coste de la tecnología, las novedades técnicas y la poca fiabilidad. Por lo tanto, la viabilidad económica depende en gran medida del coste de fabricación, su eficiencia y su fiabilidad final.

Se ha logrado equiparar la visión artificial con el empleo de la visión humana en cinco áreas de aplicación genéricas:

1. Reconocimiento: Proceso de reconocimiento de los elementos de interés, su ubicación y orientación de una escenografía completa. Esta tarea es trivial para la visión humana pero complicada en la visión artificial. Se han conseguido resultados bajo condiciones de iluminación óptimas y escenografías concretas.
2. Identificación: La finalidad de dicho proceso es la capacidad de distinguir dos objetos con la misma forma pero con objetivos diferentes. Por ejemplo, las fábricas de embotellado de *Coca-Cola Company* producen latas de *Coca Cola* y *Fanta*. Ambas latas tienen la misma forma pero sus bebidas son totalmente diferentes. Algunos métodos de identificación se han extendido ampliamente como es el caso del código de barras.

3. Exploración o rastreo: La exploración o rastreo es una de las capacidades más desarrolladas en la visión humana, pero es difícil de alcanzar en los sistemas de visión artificial. Tiene muchas aplicaciones especialmente destinadas al campo de la robótica. El sistema debe explorar un objeto en movimiento, por ejemplo en una cinta transportadora y transmitir la información sobre su trayectoria y velocidad al controlador, que determinará el movimiento de un brazo manipulado para coger la pieza en el momento y punto deseados.
4. Inspección: La inspección en lo referente a la calidad consiste en examinar y medir las características de calidad de un producto, así como sus componentes y materiales de que está elaborado. Por ejemplo, en la industria del automóvil se utiliza constantemente sistemas de visión para comprobar la correcta colocación de las piezas. Determinar la calidad de un producto es muy difícil, existen numerosas irregularidades de carácter aleatorio, obviamente, sólo detectadas por un inspector humano.
5. Medición visual: Partiendo de una luminosidad óptima y una escenografía concreta, la obtención de las dimensiones de un objeto capturado con una cámara digital es una tarea realmente sencilla. Los errores obtenidos dependerán de la resolución del sensor CMOS o CCD, distancia cámara/objeto, características de las lentes, etc.

Muchas de las técnicas de visión artificial hasta hace poco eran consideradas como prototipos de carácter experimental. Poco a poco, se están consiguiendo grandes logros en este nuevo nicho tecnológico.

## 2. Hardware versus Software.

En este punto del estado del arte se va a examinar cuál de las dos opciones es mejor a la hora de abordar un determinado problema de visión artificial.

Para realizar un tratamiento electrónico o un sistema de control de la información, a menudo es necesario poner en funcionamiento un compilador software sobre un microprocesador. Las operaciones de procesado digital de imagen requieren un cómputo matemático muy pesado, obligándonos a utilizar un microprocesador de gran potencia. Existe una gran variedad de microprocesadores comerciales, pero los más eficaces para esta tarea son los DSP.

Un procesador digital de señales o DSP “Digital Signal Processor”, es un sistema basado en un procesador o microprocesador que posee un conjunto de instrucciones, un hardware y un software muy optimizado para aplicaciones que requieran operaciones numéricas a muy alta velocidad.

Un DSP está diseñado para realizar las tareas más habituales del procesado digital: sumas, multiplicaciones y restas. Utilizan una arquitectura Harvard, donde la memoria de datos y la memoria de programa están físicamente separadas (al contrario

que la arquitectura Von Neumann, dónde los datos y programa coexisten en la misma memoria). Los elementos básicos que componen un DSP son:

- Conversores en las entradas y salidas
- Memoria de datos y memoria de programa.
- MACs: multiplicadores y acumuladores.
- ALU: Unidad aritmético-lógica.
- Registros.
- PLL: Bucles enganchados en fase.
- PWM: Módulos de control de ancho de pulso.

Sin embargo, los embotellamientos de cálculo frenan el funcionamiento del DSP impidiendo o degradando el desarrollo de procesados digitales complejos. El conflicto surge cuando los requerimientos en los tiempos de respuesta son menores que el rango factible por el DSP, incluso utilizando las técnicas de diseño software más avanzadas.

Cuando los requisitos de funcionamiento del sistema son extremadamente exigentes, la solución a diseñar excede la capacidad de los procesadores existentes. La única solución factible es el desarrollo de un hardware dedicado. Aunque esta alternativa genera resultados mucho más eficientes y eficaces, pero los costes de desarrollo son muchísimo más elevados.

Las ventajas del hardware dedicado frente el software derivan del uso de recursos necesarios para llevar a cabo una función específica. Mientras el hardware dedicado hace uso de bloques dedicados fuera de la CPU, los DSP abusan excesivamente de la CPU para realizar las tareas demandadas. En comparación con el software, el hardware dedicado proporciona un tiempo de respuesta menor, sin monopolizar los recursos de la CPU. Los microprocesadores y los DSPs hacen usos de numerosos ciclos de instrucción bloqueando la CPU.

### 3. Dispositivos lógicos programables.

Los dispositivos lógicos programables PLDs, son dispositivos que permiten la reconfiguración hardware de un circuito simplemente modificando el software de configuración, es lo contrario de la lógica cableada. Los PLDs ofrecen ventajas importantes como un corto tiempo de incorporación al mercado y una disminución significativa de costes de investigación, desarrollo, diseño y prueba de un nuevo producto. Por el contrario, los dispositivos de bajo coste poseen bajos niveles de densidad, lo que significa un mayor número de chips y menor umbral de complejidad del sistema.

Inicialmente los PLDs se orientaron al diseño de funciones hardware específicas. Realizaban funciones específicas y tenían adicionalmente una parte programable facilitando el trabajo de los diseñadores. Su capacidad de reprogramación les dio, con el tiempo, un carácter de propósito general. Estos dispositivos establecieron las bases teóricas de las actuales arquitecturas reconfigurables como las de las FPGAs.

	PLDs y FPGA	ASIC
<b>Densidad</b>	Baja/Media	Muy alta
<b>Coste de venta</b>	Medio	Bajo
<b>NRE<sup>1</sup></b>	Muy bajo	Alto
<b>Coste del diseño</b>	Bajo	Alto

Tabla 1. PLD/FPGA vs. ASIC.

## 4. Field Programmable Gate Array “FPGA”.

Una FPGA es un dispositivo semiconductor que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada mediante un lenguaje de descripción hardware (VHDL, Verilog). La lógica programable abarca desde sistemas combinacionales sencillos hasta complejos sistemas en un chip [1].

Las FPGA son resultado de la convergencia de dos tecnologías diferentes, los dispositivos lógicos programables PLDs y los circuitos integrados de aplicación específicos ASICs. Los primeros PLDs fueron los antiguos dispositivos PROM (Programmable Read-Only Memory) y se les añadió flexibilidad con las PAL (Programmable Array Logic). Poco a poco estos dispositivos han continuado creciendo en capacidad y potencia. Los ASIC, por su parte, son dispositivos potentes y de gran capacidad de integración, pero su uso siempre ha requerido una gran inversión tanto de tiempo como de dinero. En la tabla 1 se muestra una comparativa de las tecnologías configurables vs. ASICs.

Las FPGAs son el resultado de combinar la programabilidad de los PLDs (recursos de interconexión programables: fusibles, antifusibles, RAM) con las tecnologías de fabricación de los ASICs (ver figura 1).

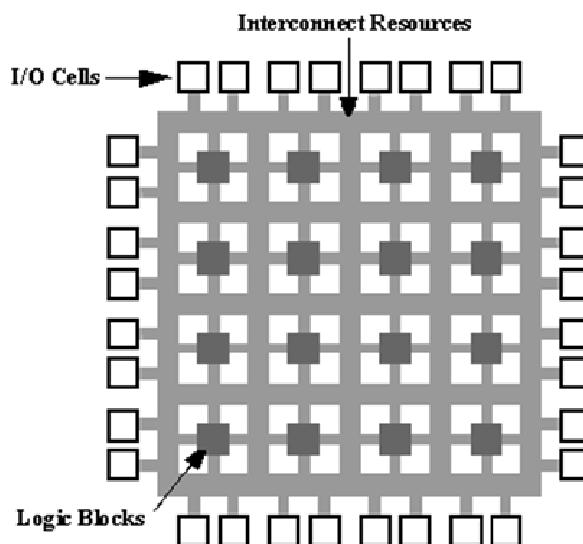


Figura 1: Arquitectura de una FPGA

<sup>1</sup> NRE: Costo único de investigación, desarrollo, diseño y prueba de un nuevo producto.

Las FPGAs de hoy en día basadas en RAM han evolucionado desde unas pocas miles de puertas en sus primeras versiones hasta un alto nivel de densidad, variable de 40k a 8M puertas.

El uso de las FPGAs en el procesado digital ha sido posible debido al avance tecnológico de estos dispositivos. Además de características generales como la capacidad de procesado en paralelo, o el número elevado de terminales de entrada/salida que facilitan la conexión de la FPGA con dispositivos externos, desde el punto de vista de la arquitectura interna, las FPGAs incluyen diferentes bloques funcionales dedicados que permiten un procesado digital muy eficiente.

#### Bloques funcionales integrados:

- A) Memorias: Memorias internas que permiten una velocidad de procesamiento mayor y pines de entrada y salida más eficientes. Las diferentes memorias son: RAM (puerto simple, puerto simple doble, bidireccional de doble puerto, etc.), ROM o registros de desplazamiento. Además de esas estructuras también es posible implementar otras como las memorias FIFO o memorias de contenido direccionable.
- B) PLLs/DLLs: Los PLLs y DLLs son utilizados para compensar los retardos de propagación del reloj a lo largo del FPGA. Otra de sus ampliaciones de interés es la multiplicación/división de frecuencia y el acondicionado del reloj (ciclo de trabajo y fase). Los DLL son bloques de bajo coste y bajo consumo, así como, muy robustos frente al ruido. Por otro lado, los PLL permiten más rango de multiplicación o división en frecuencia.
- C) Circuitos aritméticos: Algunas FPGA incorporan una gran cantidad de bloques aritméticos de baja complejidad. Por ejemplo, la familia Virtex-5 de Xilinx incluye hasta 192 multiplicadores, con los cuales se puede hacer operaciones de hasta 25x18 bits en complemento a 2. También tenemos bloques mucho más complejos como son los DPS. Por ejemplo, los dispositivos Stratix II de Altera dispone de bloques DSP con la siguiente estructura: registro + multiplicador + sumador/restador (hasta 36 bits) /acumulador (hasta 52 bits).
- D) Transceptores: En la mayoría de los dispositivos FPGA, los protocolos de comunicación se pueden implementar fácilmente haciendo uso de bloques de codificación o decodificación, serializadores o deserializadores, buffers de transmisión y recepción, entre otros.
- E) Procesadores integrados: Las FPGAs con arquitecturas más complejas de Xilinx, o Altera, incluyen procesadores embebidos y periféricos con el fin de dar soporte a soluciones basadas en SoCs. Por ejemplo, la FPGA Zynq 7000 de Xilinx incluye un procesador multinúcleo Cortex A9 de 32-bits.

## 5. Conclusión.

Actualmente el uso de DSPs está muy extendido en múltiples aplicaciones de procesado de señal, pero el principal inconveniente de esta tecnología es que, a partir de una cierta velocidad y una cierta complejidad, resultan relativamente casos y demasiados lentos. Los DSP nos limitan mucho la velocidad de proceso porque abusan excesivamente de la CPU para realizar las tareas demandadas.

En situaciones de procesado digital exigentes, no tenemos mayor alternativa que el diseño de un hardware específico. La FPGA es un excelente dispositivo para prestaciones críticas, que no son abarcables por la vía de un procesador DSP. Las FPGAs se acercan mucho más al diseño hardware, con la enorme ventaja de que se realiza por medio de un lenguaje de programación y un volcado al dispositivo, siendo posible el rediseño y la corrección de errores por reprogramación.



## ANEXO 2: Captura de imágenes

### Contenido.

1.	Introducción.....	23
2.	Partes fundamentales de una cámara fotográfica.....	23
2.1	Objetivo.....	24
2.2	Lente.....	24
2.3	Diafragma.....	24
2.4	Pentaprisma.....	25
2.5	Espejo.....	26
2.6	Ocular.....	26
2.7	Obturador.....	26
2.8	Película o sensor.....	27
3.	Tipos de cámaras digitales.....	28
3.1	Cámara CCD en blanco y negro.....	29
3.2	Cámara CMOS en blanco y negro.....	31
4.	Formas de conseguir una cámara a color.....	32
4.1	Prisma dicroico.....	32
4.2	Filtros de color.....	33
5.	Teoría de lentes.....	33
6.	Hoja de características del módulo VmodCAM.....	35

## 1. Introducción.

En este pequeño anexo realizamos un estudio de los transductores usados para captar la señal de video, es decir, las cámaras (figura 1).

Una cámara de fotos es un dispositivo utilizado, como hemos dicho anteriormente, para capturar imágenes o fotografías. Consta de una cámara oscura cerrada, con una abertura en uno de los extremos para que pueda entrar la luz, y una superficie plana donde se formará la imagen para capturar la luz. La mayoría de las cámaras poseen un objetivo formado por lentes, ubicado en la abertura para controlar la luz entrante y para enfocar la imagen, o parte de ella. El diámetro de la abertura suele modificarse con un diafragma, siendo este fijo o ajustable.

Mientras que la apertura y el brillo de la escena controlan la cantidad de luz que entra por unidad de tiempo en la cámara durante el proceso fotográfico, el obturador controla el lapso en que la luz incide en la superficie de grabación. Por ejemplo, en situación con poca luz, la velocidad de obturación será menor (mayor tiempo abierto) para permitir que la película reciba la cantidad de luz necesaria para asegurar una exposición correcta. En el caso contrario, una situación con mucha luz, la velocidad de obturación será mayor (menor tiempo abierto) pues no necesitamos tanta cantidad de luz para conseguir una exposición correcta.



Figura 1: Cámara digital doméstica.

## 2. Partes fundamentales de una cámara fotográfica.

La figura 2 muestra las partes más importantes de una cámara fotográfica. A continuación se analiza cada una de ellas.

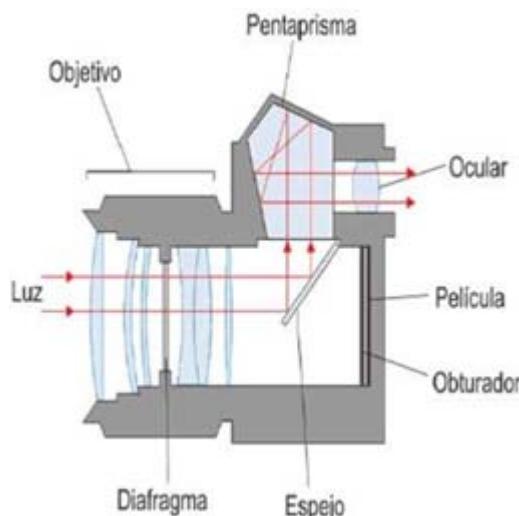


Figura 2: Partes fundamentales de una cámara fotográfica.

## 2.1 Objetivo.

Se denomina objetivo (figura 3) al dispositivo que contiene un juego de lentes convergentes y divergentes, así como el diafragma, que forman la parte óptica de una cámara tanto fotográfica como de vídeo.



Figura 3. Objetivo de una cámara.

### b. Lente.

El lente es el componente de la cámara que sirve para enfocar y regular el foco (por ejemplo, las cámaras con *zoom* son capaces de acercar y alejar la imagen). Captan los haces de luz y los ordenan en el otro lado de forma convergente (figura 4).

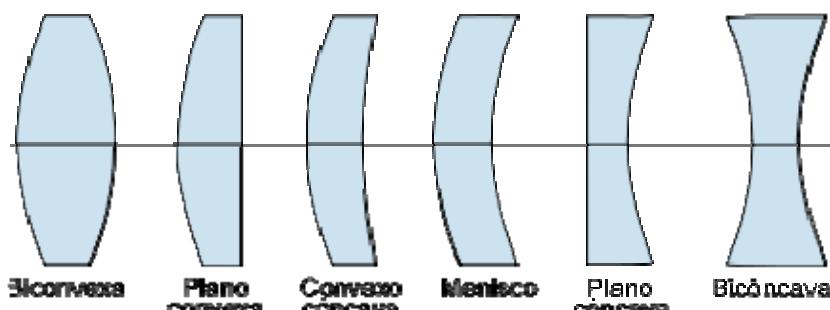


Figura 4. Tipos de lentes.

La imagen a captar, puede estar en el foco o no. Si está en el foco, la imagen está formada por puntos uno al lado del otro, lo que permite una imagen limpia y nítida. Si esta fuera del foco, la imagen está formada por círculos, ya que los haces de luz llegan al material sensible más separados.

Al girar el objetivo se enfoca la imagen en el ocular de tal forma que en la película se forme una imagen nítida.

### c. Diafragma.

El diafragma entre las lentes anteriores y posteriores controla la cantidad de luz que penetra en la cámara. Como se indicó anteriormente, para fotografiar un escenario muy oscuro se necesita una abertura mayor, mientras que para escenarios muy luminosos una abertura menor. Por lo tanto, la abertura del diafragma establecerá la nitidez del objeto enfocado.



Figura 5. Notación de distancia focal y abertura de diafragma en un objetivo comercial.

La letra “*F*” seguido del número que corresponda representa el cociente entre la distancia focal y el diámetro de la abertura del objetivo. Se habla de una abertura pequeña cuando nos referimos a un número “*F*” mayor, y a una abertura grande a un número “*F*” menor (ver figuras 5 y 6).

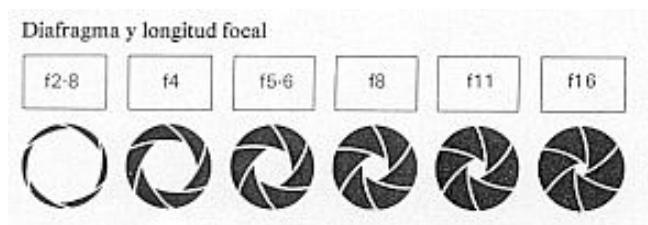


Figura 6. Relación entre la distancia focal y el diámetro de apertura.

El diafragma está formado por un conjunto de laminillas que se abren o se cierran determinando que cantidad de luz va a recibir la película.

#### d. Pentaprisma.

Un pentaprisma es un prisma reflectivo de cinco caras empleado para desviar un rayo de luz en ángulo de 90° (figura 7). El pentaprisma refleja la luz desde la pantalla de enfoque hasta el ocular para poder ver la fotografía que se va a realizar.

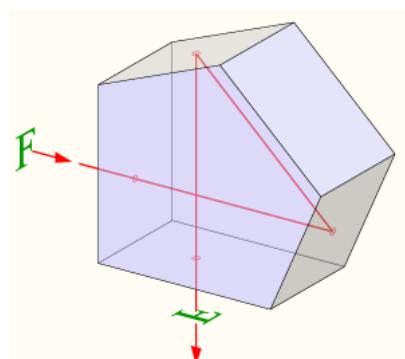


Figura 7: Pentaprisma.

## e. Espejo.

El espejo realiza dos funciones fundamentales: protege la película de la luz que pasa por el lente y permite la proyección, con el pentaprisma, en el visor de la imagen enfocada (figura 8).



Figura 8. Espejo de una cámara.

Por lo tanto, cuando se realiza un disparo para sacar una foto, el espejo se levanta y deja que la luz pase para la película permitiendo que se forme la imagen en ella.

## f. Ocular.

Permite al fotógrafo ver la imagen y disparar simultáneamente (a diferencia de las grandes cámaras antiguas, no es necesario quitar el espejo durante la toma de la fotografía) (figura 9).



Figura 9. Ocular de una cámara.

## g. Obturador.

El obturador (figura 10) es el dispositivo que controla el tiempo durante el que llega la luz al dispositivo fotosensible (*película* en la fotografía química o *sensor* en la fotografía digital). Este tiempo es conocido como la *velocidad de obturación*, y de él se desprenden conceptos como el congelado o el barrido fotográfico. Junto con la apertura del diafragma, la velocidad de obturación es el principal mecanismo para controlar la cantidad de luz que llega al elemento fotosensible.



Figura 10: Obturador de una cámara.

Cuando disparamos la cámara, el obturador se abre, tomando la imagen del objetivo en el sensor o en la película.

## h. Película o sensor.

La cámara utilizará una *película* o un *sensor* para grabar la foto dependiendo de si es una cámara analógica o digital.



Figura 11: Carrete fotográfico.

En el caso de las cámaras analógicas, se utilizan películas fotográficas. La película fotográfica es una superficie transparente, de plástico y casi siempre flexible. Está hecha de *acetato de celulosa* u otros plásticos y recubierta de una delgada capa de emulsión fotográfica, formada por una sustancia sensible a la luz, como *el bromuro de plata*. Es obvio, por las propiedades de la película, que para obtener una imagen inalterable en futuras exposiciones a la luz necesitamos aplicar a la película una serie de procesos químicos. Este proceso es llamado revelado fotográfico y básicamente reduce la sensibilidad a la luz de la emulsión y estabiliza la imagen (ver figura 11).

En el caso de las cámaras digitales, se utilizan sensores electrónicos (ver figura 12). El sensor de imagen capta la luz que compone la imagen y la convierte en una señal, tanto analógica como digital. Se trata de un pequeño chip formado por millones de componentes sensibles a la luz, como pueden ser, los fotodiodos o los fototransistores. Al estar expuestos a la luz capturada por el objetivo cada uno de los componentes genera una respuesta eléctrica que formará la imagen.

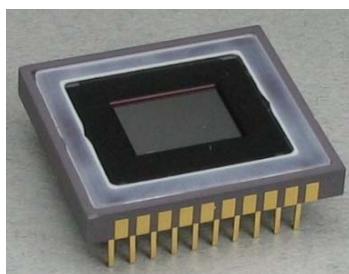


Figura12: Sensor CMOS

El sensor es una matriz de elementos fotosensibles que convierte la luz captada en señales eléctricas que posteriormente pueden ser convertidas, analizadas y almacenadas. El valor obtenido será representado analógicamente (como una señal de barrido o escaneo) o digitalmente (mediante muestreo y conversión).

Finalmente, el fichero informático generado podrá ser visualizado en un monitor evitando la necesidad del papel fotográfico, obteniendo una imagen digital o un vídeo (sucesión continua de imágenes a alta velocidad).

### 3. Tipos de cámaras digitales.

Empezaremos con un breve estudio de los sistemas en blanco y negro, todavía utilizados en aplicaciones de vigilancia, para después centrarnos en los sistemas a color. En el apartado anterior, se ha explicado básicamente que todas las cámaras necesitan una parte óptica que forme una imagen en el sensor, sensible a la luz como el cristalino del ojo. Los diferentes tipos de elemento sensible dan lugar a diferentes tipos de cámaras.

Como trabajaremos con una cámara digital, es esencial, conocer la unidad mínima de visualización, el píxel. Un píxel es la menor unidad homogénea en color que forma una parte de una imagen digital, utilizada en la fotografía o en un fotograma de un vídeo. (ver figura 13).

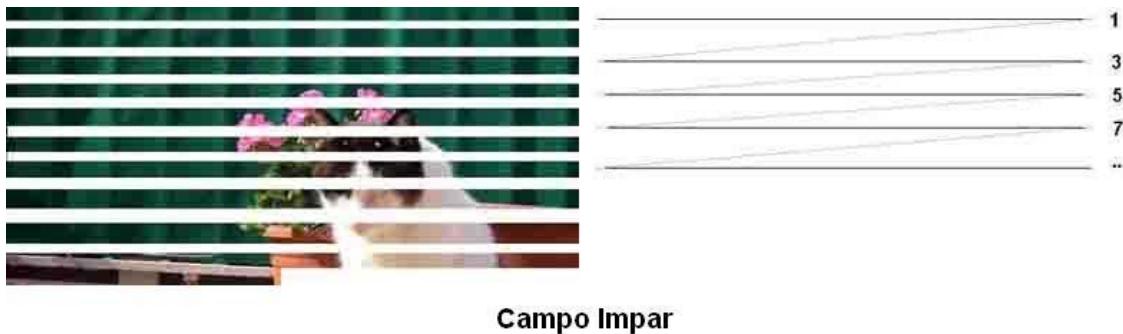


Figura 13: Imagen “píxelada”.

Para formar una imagen los píxeles deben ser ordenados a lo largo de la pantalla. En las primeras cámaras se realizaba una ordenación por líneas horizontales que se conserva todavía hoy.

Existen dos tipos de barridos en la imagen: barrido entrelazado y barrido progresivo. El barrido entrelazado es un sistema de captación y representación de

imágenes utilizado en la televisión para evitar el parpadeo o “flicker”. Consiste en captar y representar primero las líneas impares y después las líneas pares. Como consecuencia del entrelazado, cada imagen supone dos recorridos verticales de pantalla (ver figura 14). El barrido progresivo, como ya dice la palabra, capta y representa las líneas todas en orden, es un método de exploración secuencial. (ver figura 15).



Campo Impar

Figura 14: Barrido entrelazado.

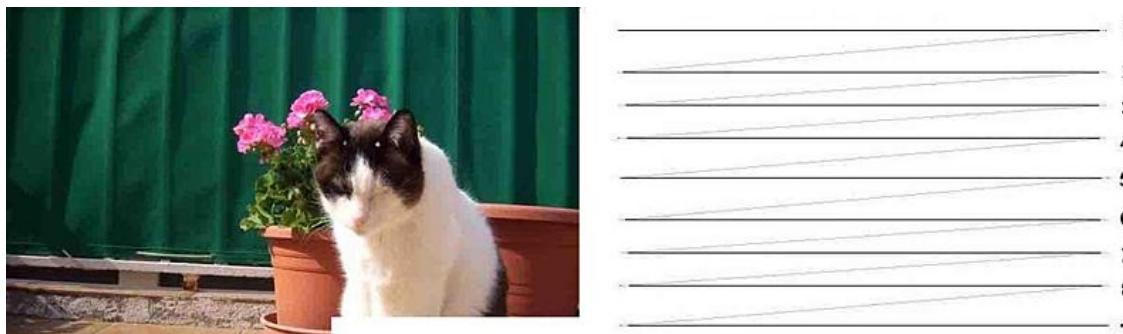


Figura 15: Barrido secuencial.

### a. Cámara CCD en blanco y negro.

Un dispositivo de carga acoplada del inglés “charge-coupled device” es un chip que contiene un número muy grande de condensadores (realmente son transistores de efecto de campo con capacidad parásita). Estos componentes son sensibles a la luz, por lo tanto, cuando los fotones inciden sobre un componente la capacidad parásita del transistor almacena una carga directamente proporcional a la intensidad de luz incidente (ver figura 16).

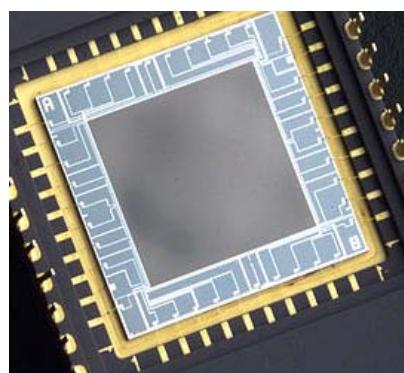


Figura 16: Sensor CCD.

Mediante un circuito de control interno, cada condensador puede transferir su carga eléctrica a uno o a varios de los condensadores que estén a su lado en el circuito integrado (figura 17). Los chips CCD, tienen estructura matricial y se leen desplazando la carga por filas. Existe una fila no expuesta a la luz llamada fila de lectura que almacena los niveles de luz captados. Como hemos comentado en la introducción de las cámaras, la lectura se puede hacer de manera progresiva o entrelazada.

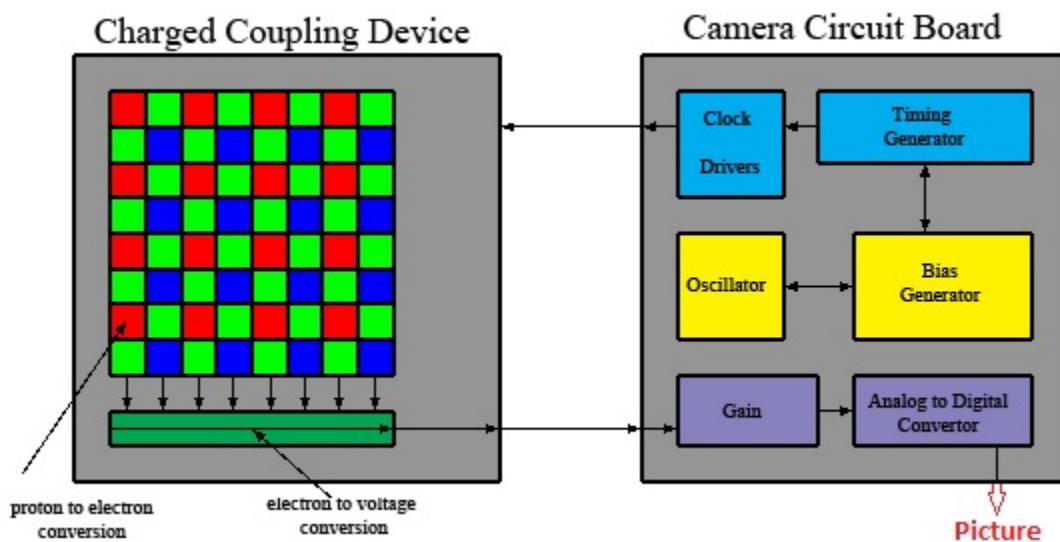


Figura 17: Circuito de control interno de un sensor CCD.

El tiempo de carga es fundamental a la hora de querer tomar una imagen de una manera u otra. Supondremos que el tiempo de lectura es despreciable frente al tiempo de carga. Cuando sacamos una foto es necesario ajustar el tiempo del obturador, para poder capturar imágenes en movimiento (obturación corta) e imágenes con poca luz (obturación larga).

Uno de los problemas más comunes de los sensores CCD es su facilidad para producir el efecto "smear". El efecto "smear" exagera las zonas brillantes de la foto. La causa de este efecto tan molesto es que, en ocasiones, si el sensor almacena una carga muy elevada en una línea, luego no es capaz de desplazarla totalmente en un ciclo de lectura. Por lo tanto, las cargas que no fueron trasladadas en el primer

momento se suman a las que llegan a esa posición (provenientes de otros píxeles de la misma vertical que se están desplazando hacia la línea de lectura). El efecto resultante es un exceso de brillo en determinadas líneas de la imagen (ver figura 18).

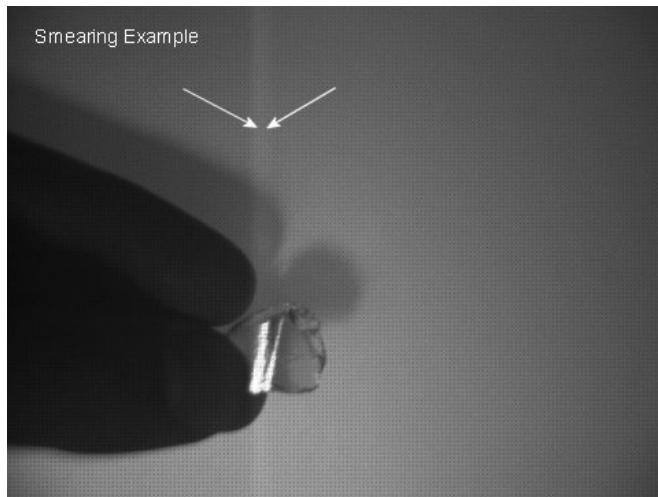


Figura 18: Efecto “smear” en una fotografía.

## b. Cámara CMOS en blanco y negro.

*Complementary metal-oxide-semiconductor* o CMOS (semiconductor complementario de óxido metálico) es una de las tecnologías empleadas en la fabricación de circuitos integrados.

Los sensores CMOS, al igual que los CCD, se basan en el efecto fotoeléctrico (figura 19). La diferencia fundamental con el CCD es que es posible la lectura individual de los píxeles. Por lo tanto, los píxeles son realmente independientes entre ellos, evitando de esta manera, el uso de desplazamientos. Además, la independencia entre píxeles permite obtener imágenes a mayor velocidad y sin el problema del “smear” asociado a los sensores CCD.



Figura 19: Cámara CMOS

El principal inconveniente de esta tecnología es que posee un menor rango dinámico, perdiendo un gran contraste. Además, la densidad de integración es mucho menor que en el caso de los CCD.

## 4. Formas de conseguir una cámara a color.

### a. Prisma dicroico.

El prisma dicroico se utiliza en cámaras a color de muy altas prestaciones y coste muy elevado.

Un prisma dicroico es un material capaz de dividir un haz de luz policromática en diversos haces monocromáticos con distintas longitudes de onda (ver figura 20).

Estos tipos de filtros se fabrican depositando en el vacío capas de sustancias reflectivas sobre un sustrato, generalmente de vidrio. Variando el número y grosor de estas capas se puede ajustar el filtro a la frecuencia y ancho de banda deseado.

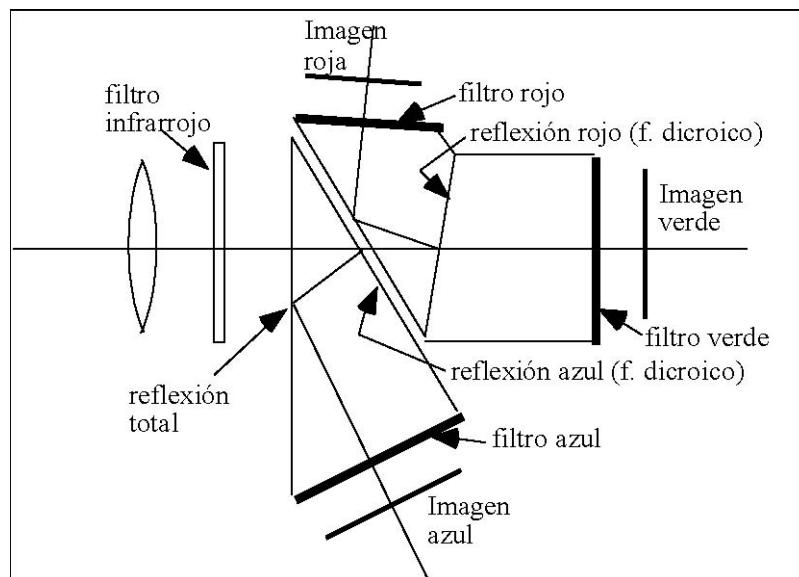


Figura 20: Estructura de un prisma dicroico

El prisma dicroico se basa en tres prismas y dos filtros dicroicos (que refleja un color y deja pasar los demás).

1. El primer prisma refleja la luz azul, volviendo a la primera cara del prisma con un ángulo tal que se produce reflexión total (ángulo de incidencia mayor que el ángulo crítico). La luz azul se hace pasar por un último filtro, transparente y normal, para asegurar que no entra con otras frecuencias distintas a la del azul.
2. Al segundo prisma llega una imagen con las componentes rojas y verdes. La luz roja es reflejada y filtrada.
3. Al último prisma sólo le llega la componente verde. Es filtrado por seguridad.

El dicroico está diseñado de tal manera que el camino óptico de las tres componentes sea de igual longitud y así evitar el desfase entre las diferentes componentes.

Una vez descompuesta la luz en las componentes rojo, verde y azul, cada componente de color es muestreada por cada uno de sus respectivos sensores (por ejemplo, un sensor CCD en el caso de los “*triple CCD*”).

## b. Filtros de color.

La inmensa mayoría de las cámaras a color no disponen de prisma dicroico sino que utilizan filtros de color para generar una imagen. Se emplea un único sensor al que se le coloca un filtro de color delante de cada uno de sus elementos fotosensibles. De esta forma cada píxel sólo obtiene un color primario y la imagen a color obtenida está formada mediante la interpolación de cada uno de los valores de cada píxel (ver figura 21).

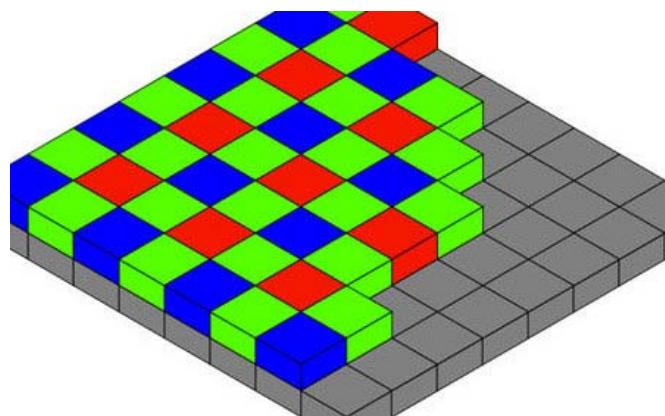


Figura 21: Filtro Bayer

## 5. Teoría de lentes.

Las lentes son los elementos ópticos que permiten recoger la luz de una escena y enfocarla sobre el sensor o película de la cámara. Las lentes pueden ser convergentes o divergentes.

La lente convergente (o positiva) es más gruesa en su parte central y más estrecha en sus bordes. Concentran (hacen converger) en un punto los rayos de luz que las atraviesan. A este punto se le denomina foco y la separación entre él y la lente se conoce como distancia focal (ver figura 22).

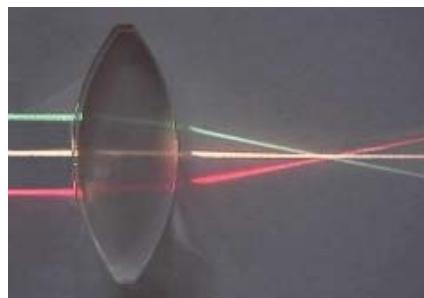


Figura 22: Lente convergente

Las personas con hipermetropía no ven bien de cerca y tienen que alejarse de los objetos para verlos. Una posible corrección de la hipermetropía es el uso de lentes convergentes (figura 23).

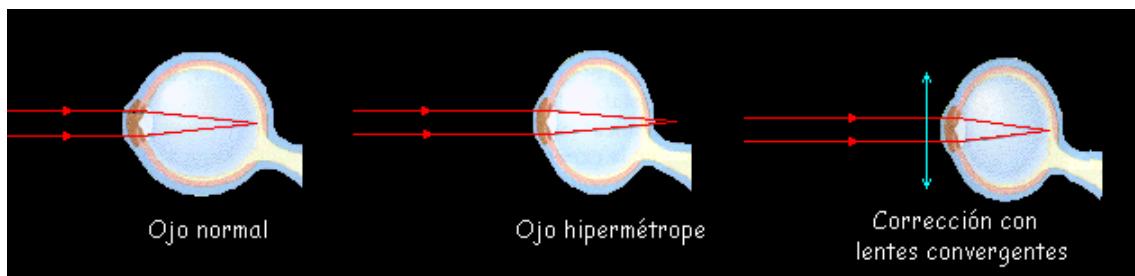


Figura 23: Corrección de la hipermetropía

La lente divergente (o negativa) es más gruesa en sus bordes y más estrecha en su parte central. Separan (hacen divergir) los rayos de luz que pasan por ellas (ver figura 24).

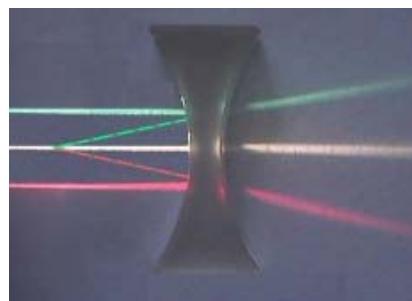


Figura 24: Lente divergente

Los miopes no ven bien de lejos y tienden a acercarse demasiado a los objetos para lograr verlos. Las lentes divergentes sirven para corregir este defecto (figura 25).

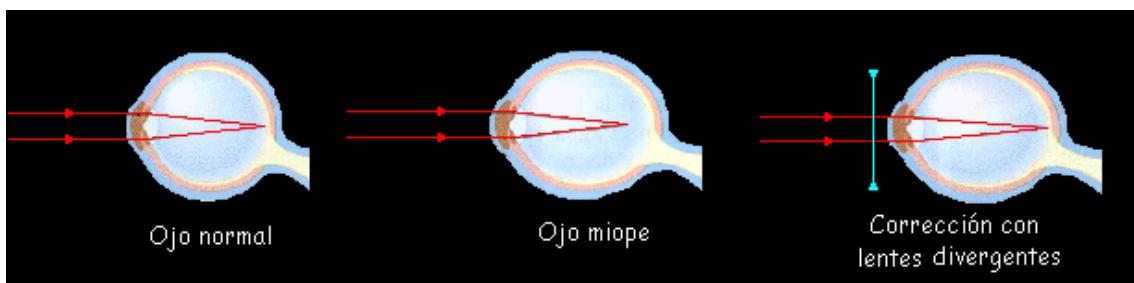


Figura 25: Corrección de la miopía

## 6. Hoja de características del módulo VmodCAM.

La placa *VmodCAM* de *Digilent* [2] permite la adquisición de imágenes y video digital. Puede conectarse a cualquier placa de desarrollo de FPGAs de este fabricante, siempre y cuando, la placa posea un conector VHDCI. Cuenta con dos sensores CMOS *Aptina MT9D112* de 2 megapíxeles, con lo que es posible obtener una resolución de hasta 1600x1200 píxeles y una velocidad de muestreo de 15FPS, dependiendo de la configuración utilizada (ver figura 26).

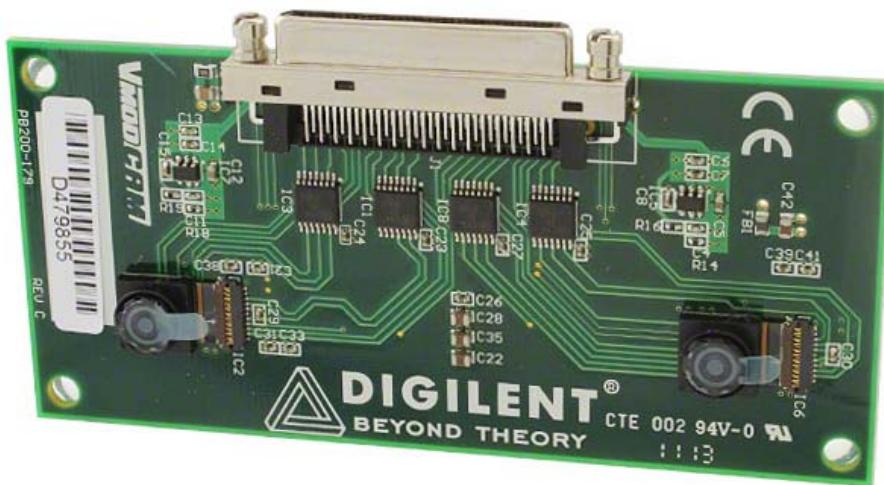


Figura 26: Placa VmodCAM de Digilent.

Su diseño *System-On-a-Chip* integra un procesador que permite seleccionar formatos de salida, escalas y efectos de preprocessado. Los datos de salida son enviados en paralelo por el conector VHDCI en formato YCrCB, RGB o ROW. Para más información consultar la hoja de características especificada por el fabricante *Digilent* “*VmodCAM Reference Manual*”.



## ANEXO 3: Marco teórico.

### Contenido.

1. Introducción .....	38
2. Tipos de imágenes digitales .....	38
2.1    Mapa de bits.....	38
2.2    Imágenes vectoriales .....	39
3. Concepto del color.....	40
3.1    Modelo RGB.....	40
3.2    Modelo CMYK.....	41
3.3    Modelo HSL.....	42
3.4    Modelo HSV.....	44
3.5    Conclusión sobre el modelo de color a utilizar en el proyecto. ....	45
4. Procesado de imágenes.....	45
4.1    Técnicas asociadas al procesado de imágenes.....	46
4.1.1    Histograma.....	46
4.1.2    Detección de bordes.....	49
4.2    Conclusión sobre el modelo de procesado de imagen a utilizar en el proyecto. ....	58
5. Distancias entre la imagen de referencia y la imagen muestreada.....	58
5.1    Métricas utilizadas para el cálculo de distancias.....	59
5.1.1    Correlación.....	59
5.1.2    Prueba $\chi^2$ .....	60
5.1.3    Intersección.....	60
5.1.4    Método de Bhattacharyya. ....	60
5.2    Conclusión sobre la métrica a utilizar para el cálculo de distancias. ....	61
6. Conclusión sobre la arquitectura de sistema propuesta en el proyecto. ....	61

## 7. Introducción.

En este anexo se recogen los conceptos referentes al procesado de imagen estudiados para poder llevar a cabo el proyecto. Partiendo de un conocimiento muy básico del procesado digital de imágenes se ha indagado sobre los diferentes puntos teóricos que envuelven estas técnicas [3][4][5].

## 8. Tipos de imágenes digitales

### a. Mapa de bits

Existen dos formas de representar las imágenes digitales, mediante un gráfico vectorial o mediante un mapa de bits, dependiendo de si poseen una resolución estática o dinámica.

Una imagen en mapa de bits es un fichero constituido mediante una gran cantidad de píxeles donde cada píxel representa un color. La matriz obtenida mediante la integración total de cada uno de dichos píxeles representa la imagen (figura 1).

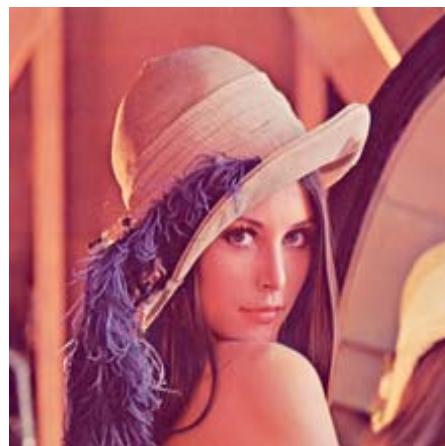


Figura 1: Lenna, imagen de prueba para los algoritmos de compresión de imagen.

La calidad de la imagen será definida por su altura y anchura (en píxeles X\*Y) y su profundidad de color (bits de representación por píxel). La primera característica determina el tamaño de la imagen y la segunda la calidad del color de la imagen. Cuanto más rango de colores mayor calidad, es decir, cuantos más bits de representación mayor calidad.

Cuando hacemos *zoom* sobre una foto no hacemos nada más que un rescalado. La reducción de la calidad de la imagen es severamente drástica y podemos percibir la individualidad de los píxeles. A este efecto se denomina *pixelado* (ver figura 2).



Figura 2: Imagen pixelado.

## b. Imágenes vectoriales

Una imagen vectorial es una imagen digital compuesta por entidades geométricas simples e independientes entre ellas. Por lo tanto, cada una de las entidades que componen la imagen está definida mediante un modelo matemático (coordenadas, grosor, color del contorno, color del relleno, etc).

La principal ventaja de este tipo de imágenes frente al mapa de bits es que se puede ampliar el tamaño de una imagen tanto como se quiera sin sufrir pérdida de calidad. (Ver figura 3).

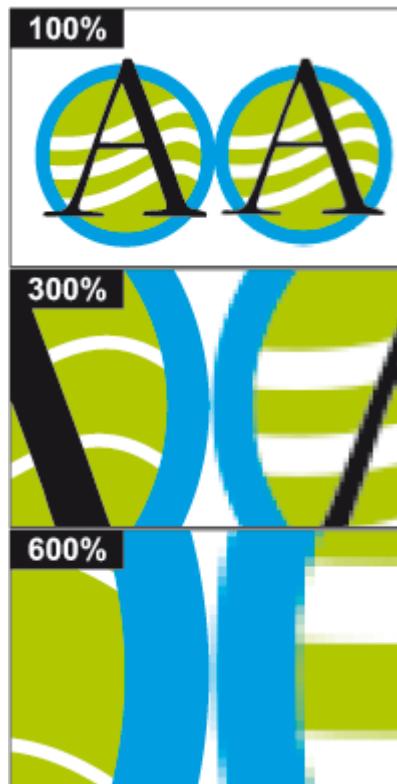


Figura 3: Mapa de bits versus imagen vectorial.

## 9. Concepto del color.

El color es la interpretación individual y subjetiva de las ondas electromagnéticas que refleja un objeto iluminado. Las ondas reflejadas son recogidas por el ojo humano que actúa como transductor, transformando las longitudes de onda de la luz en las distintas señales nerviosas.

El ser humano posee un espectro electromagnético muy limitado capaz de percibir longitudes de ondas desde los 380nm hasta los 780nm. Esta región del espectro electromagnético es denominada espectro visible.



Figura 4: Espectro visible del ojo humano.

Dentro de este espectro visible se pueden clasificar 6 colores, incluyendo en sus límites todas las demás tonalidades. (Ver figura 4)

Otro punto a tener en cuenta, es que el ojo humano solo percibe las diferentes longitudes de onda cuando existe iluminación, por lo tanto, en ausencia de ella, solo vemos en blanco o negro.

### a. Modelo RGB.

El modelo RGB (del inglés Red, Green and Blue) es uno de los espacios de color más utilizado.

RGB es un modelo de color aditivo, con el que es posible la representación de cualquier color mediante la mezcla por adición de los tres colores primarios de la luz: rojo, verde y azul (los percibidos por los conos en el ojo humano). Se dice que es un sistema de color aditivo porque la suma de todos sus componentes, *Red + Green + Blue*, genera el color blanco. (Ver figura 5).

El modelo RGB es muy utilizado para representar imágenes que serán mostradas o impresas utilizando los sistemas electrónicos disponibles en la actualidad. Por su simplicidad, también es el modelo más usado en programas de fotografía como el *Photoshop* o el *Gimp*.

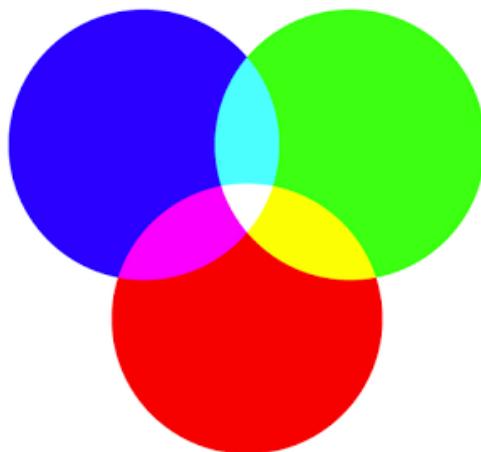


Figura 5: Modelo de color RGB.

Cuando se utiliza el modelo RGB cada píxel se representa mediante un valor digital en el que parte de los bits corresponden al color rojo, otra parte al verde y otra al azul. Existen diferentes estándares que definen el número de bits asociado a cada componente de color. El más habitual es el RGB888, en el que se utilizan 8 bits para cada color. De esta forma cada pixel se representa mediante 24 bits y la intensidad de cada una de las componentes de color del píxel (R, G y B) oscila entre 0 “negro” y 255 “blanco”. El rango de colores final se obtiene mediante las diferentes combinaciones de rojo, verde y azul.

Algunos ejemplos:

- Rojo: R = 255, G = 0 , B = 0
- Verde: R = 0, G = 255 , B = 0
- Azul: R = 0, G = 0 , B = 255
- Negro: R = 0, G = 0 , B = 0
- Blanco: R = 255, G = 255, B = 255

Con una resolución de 24 bits por píxel podemos obtener 17,6 millones de colores.

Otros estándares son el RGB555, en el que cada componente R, G y B se representa mediante una combinación de 5 bits (de “0” a “31”), o el RGB565, en el que a las componentes R, G y B le corresponden 5, 6 y 5 bits respectivamente. El formato RGB565 es el que utiliza el módulo *VmodCAM* de *Digilent*, es decir, las cámaras utilizadas en este proyecto.

## b. Modelo CMYK.

El modelo CMYK es el modelo opuesto al modelo RGB. CMYK es un modelo de color subtractivo que permite crear una amplia gama de colores (ver figura 6).

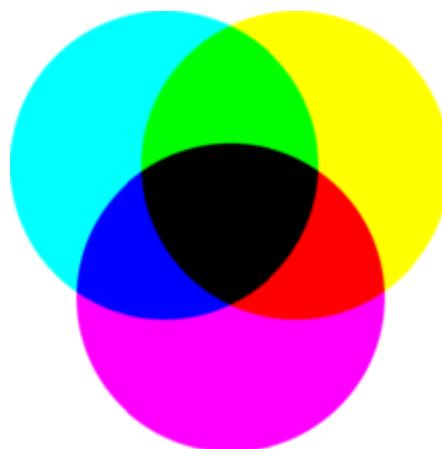


Figura 6: Modelo de color CMYK

Está formado por los siguientes colores:

- C = Cian “Cyan”.
- M = Magenta “Magenta”.
- Y = Amarillo “Yellow”.
- K = Negro “Black o Key”.

Por lo tanto, CMYK está compuesto por los colores complementarios (cian, magenta y amarillo) a los 3 primarios (rojo, verde y azul):

- Cian: verde + azul – rojo.
- Magenta: rojo + azul – verde.
- Amarillo: rojo + verde – azul.

El modelo CMYK tiene mucho éxito en la industria de la impresión. El color negro puede ser producido mediante la mezcla de todos los colores (cian, magenta y amarillo).

### c. Modelo HSL.



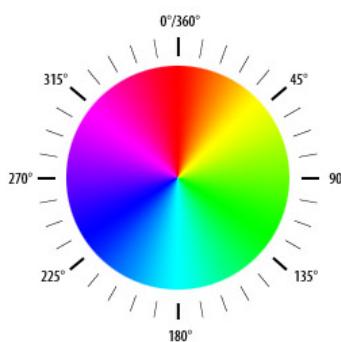
Figura 7: Componentes del modelo de color HSL

El modelo **HSL** (del inglés *Hue, Saturation and Lightness*), define un píxel en términos de sus componentes constituyentes, es decir, en función del tono (o matiz), la claridad (o luminosidad) y la saturación (figura 7).

Para especificar un color se selecciona un tono a partir del espectro del arco iris, y se establece su saturación (pureza del color) y su luminosidad (de claro a oscuro).

#### Tono (Hue):

El tono se calcula en función de la longitud de onda dominante y se representa en grados, del 0 al 360. (Ver figura 8).



**Figura 8: Tono**

#### Saturación (Saturation):

La saturación indica la pureza del color. Proporción de la luz pura de la longitud de onda dominante y la luz blanca necesaria para definir el color. Un color 100% puro tiene una saturación del 100% y no contiene luz blanca. Los colores que se alejan del color puro contienen una mezcla entre luz blanca y el color puro comprendida entre 0% y el 100%. El gris tiene un 0% de saturación ya que no contienen colores.

#### Luminosidad (Lightness):

La luminosidad u oscuridad relativa del color normalmente se expresa como un porcentaje comprendido entre 0% (negro) y 100% (blanco).

En conclusión, este modelo de color es más cercano a la idea humana del color al añadir términos de tono y saturación. La representación gráfica es un doble cono, en cuyos vértices se localizan el negro y el blanco. El centro del doble cono sería, por lo tanto, un tono gris dado que el eje que une sus vértices indica la luminancia. (Ver figura 9).

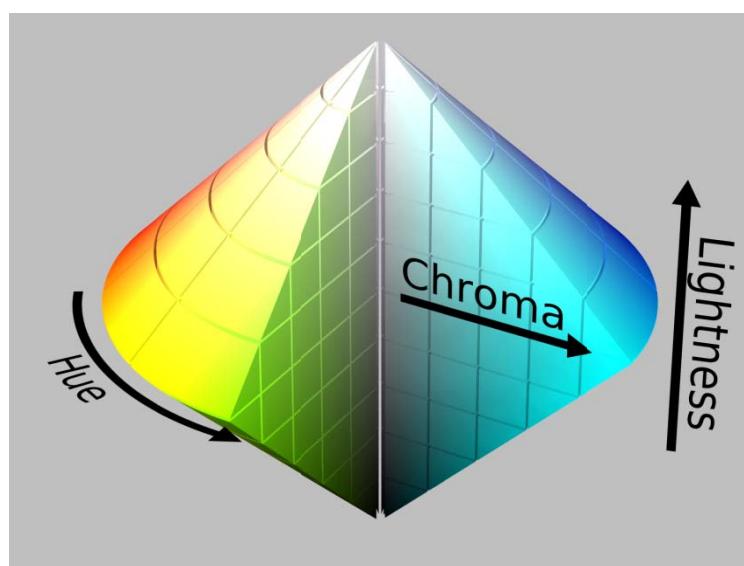


Figura 9: Representación gráfica del modelo HSL

#### d. Modelo HSV.

El modelo HSV es prácticamente igual que el modelo HLS. La diferencia radica en la substitución del parámetro *Lightness* por *Value*. Luminosidad y valor representan lo mismo, la única diferencia es que la forma generada por HLV es un cono, por lo tanto, el vértice representa el color negro y la base el color blanco. (ver figura 10).

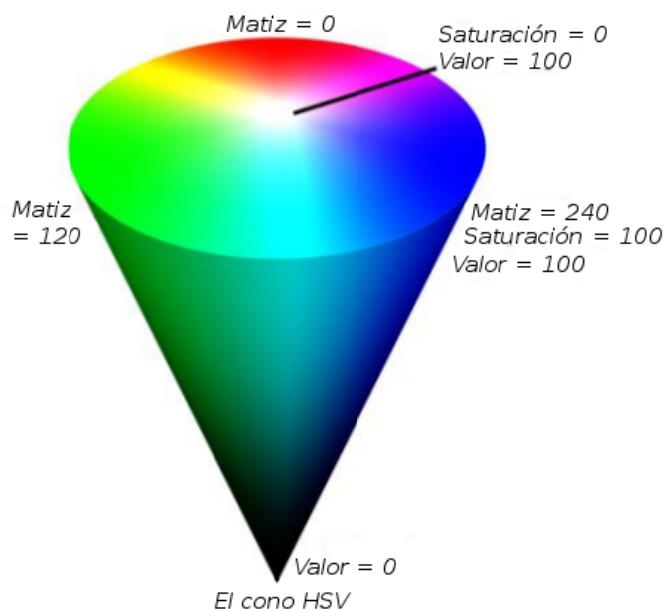


Figura 10: Representación gráfica del modelo HSV.

## e. Conclusión sobre el modelo de color a utilizar en el proyecto.

Los modelos HSV o HSL se utilizan con mucha frecuencia en aplicaciones de visión artificial y en el procesado digital de imagen para la identificación de figuras o la segmentación de imágenes.

En la mayor parte de los algoritmos de visión artificial las imágenes a color son transformadas a escala de grises y posteriormente se les aplica diferentes técnicas de procesado. El modelo RGB depende fuertemente de la luminosidad que llegue al objeto, por lo tanto, obliga a poner la cámara en una situación extremadamente idónea donde la iluminación permanezca constante.

Los modelos HSV o HSL, por el contrario, extraen la componente lumínica de la imagen, desligándonos de dicha dependencia. Por lo tanto, podemos tener una escenografía mucho más complicada (peor iluminada) y conseguir identificar objetos de manera más eficaz e eficiente.

Teniendo en cuenta las consideraciones anteriores, para la realización de este proyecto se propone, como paso previo al procesado de una imagen, transformar la imagen proporcionada por el módulo VmodCAM (modelo RGB565) a un modelo HSL.

## 10. Procesado de imágenes.

El trabajo propuesto en este proyecto tiene como finalidad la identificación de imágenes en un escenario mediante visión artificial. Para ello, el sistema a desarrollar debe almacenar en una memoria una imagen de referencia de un objeto, y posteriormente, el algoritmo de procesado debe ir analizando los escenarios y objetos que filma para obtener el grado de similitud con la imagen de referencia almacenada en la memoria.

El sistema consta de tres etapas: adquisición, procesado e interpretación. (Ver figura 11).

### Adquisición de datos:

La función de la primera etapa es la adquisición de imágenes del escenario que monitoriza la cámara. La cámara de video toma una imagen de referencia y las posteriores imágenes son comparadas con la imagen almacenada en memoria.

### *ANEXO: Captura de Imágenes*

### Procesado:

Durante esta fase procederemos con el procesado de las imágenes tomadas por la cámara. Se hará un prototipo mediante *matlab*, y una vez verificado el funcionamiento, implementaremos el algoritmo completo en una FPGA.

Usaremos el modelo de color HSL (Hue, Saturation and Lightness) para tratar de minimizar la dependencia con la iluminación escénica.

Se realizará, una vez obtenido el modelo de color HSL, un algoritmo de identificación de objetos. Durante las siguientes páginas explicaremos los diferentes métodos y alternativas para realizar el algoritmo.

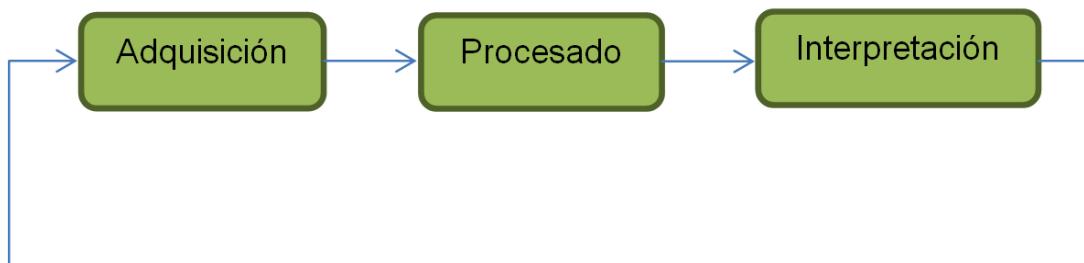


Figura 21: Etapas del procesado de imágenes.

#### Interpretación:

Expresamos el grado de similitud entre la imagen de referencia almacenada en la memoria y las imágenes capturadas por la cámara.

### 4.1 Técnicas asociadas al procesado de imágenes.

El procesado digital de imagen es el conjunto de técnicas que se aplican en las imágenes digitales con el objetivo de modificar o buscar cierta información en ellas.

En consecuencia, se pueden obtener dos resultados: una nueva imagen a partir de la modificación de una imagen original o información sobre ciertos datos de una imagen.

Existen multitud de técnicas asociadas al procesado de imágenes, en los subapartados siguientes introduciremos algunas de las técnicas analizadas durante la realización de este proyecto. Si bien, no todas ellas fueron finalmente implementadas.

#### 4.1.1 Histograma.

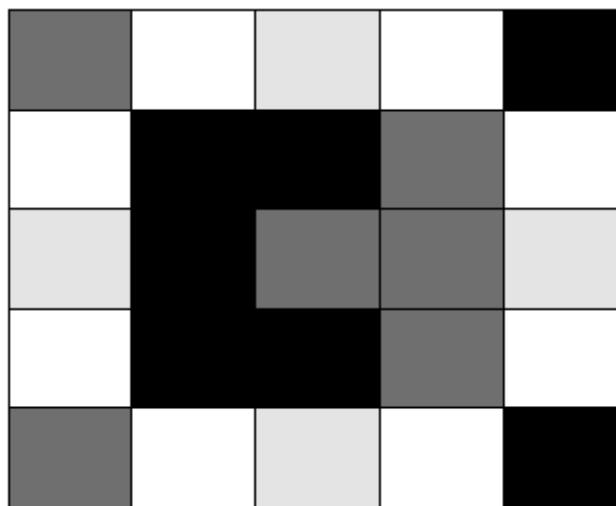
Un histograma es una representación gráfica de una variable en forma de barras, donde la superficie de cada barra corresponde a la frecuencia de los valores representados [6].

Son utilizados para obtener una visión general de la distribución de elementos de una población respecto a una característica de interés para el usuario. Podemos caracterizar de esta manera comportamientos observando el grado de homogeneidad, variabilidad y, por supuesto, evidenciar si no hay ninguna tendencia.

¿Cuando vemos una imagen realmente podemos decir de manera exacta si una imagen tiene más brillo que otra? No, es prácticamente imposible. Aparte de ser algo muy subjetivo cada foto tiene unos valores diferentes de brillo o luminosidad, que proporcionan autenticidad respecto a las demás.

Este apartado de la memoria establecerá las pautas más básicas para el estudio de la técnica del *Histograma* para la diferenciación entre imágenes.

Utilizaremos la figura 12, como ejemplo para el estudio de esta técnica.



**Figura 12: Imagen utilizada para la realización del histograma.**

Se trata de una imagen en escala de grises, 5x5, cuyos valores representan intensidades de gris. Como podemos apreciar a simple vista, tenemos 4 tipos de colores: Negro, gris oscuro, gris claro y blanco. Por consecuencia, para la codificación de la imagen utilizaremos 2 bits.

Color	Código	Píxeles
Negro	00	7
Gris oscuro	01	6
Gris claro	10	4
Blanco	11	8
<b>Total</b>	-	<b>25</b>

El número asignado a cada color corresponde de mayor a menor según la intensidad de brillo, es decir, al negro que es el valor más oscuro se le asigna el número 0, mientras que el blanco que es el valor más brillante se le asigna, como es obvio, el número más alto (el número 3 en este caso).

La figura 13 muestra el *histograma* asociado a la imagen de la figura 12 anteriormente mostrada.

## Píxeles de la figura X

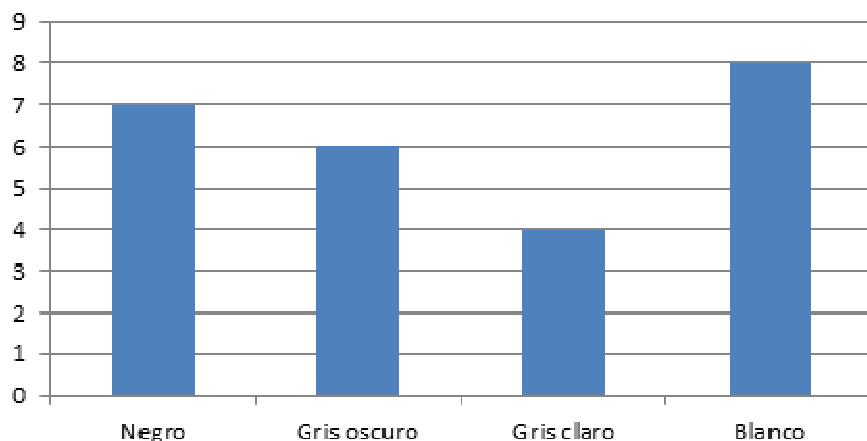


Figura 13: Histograma de la figura 12.

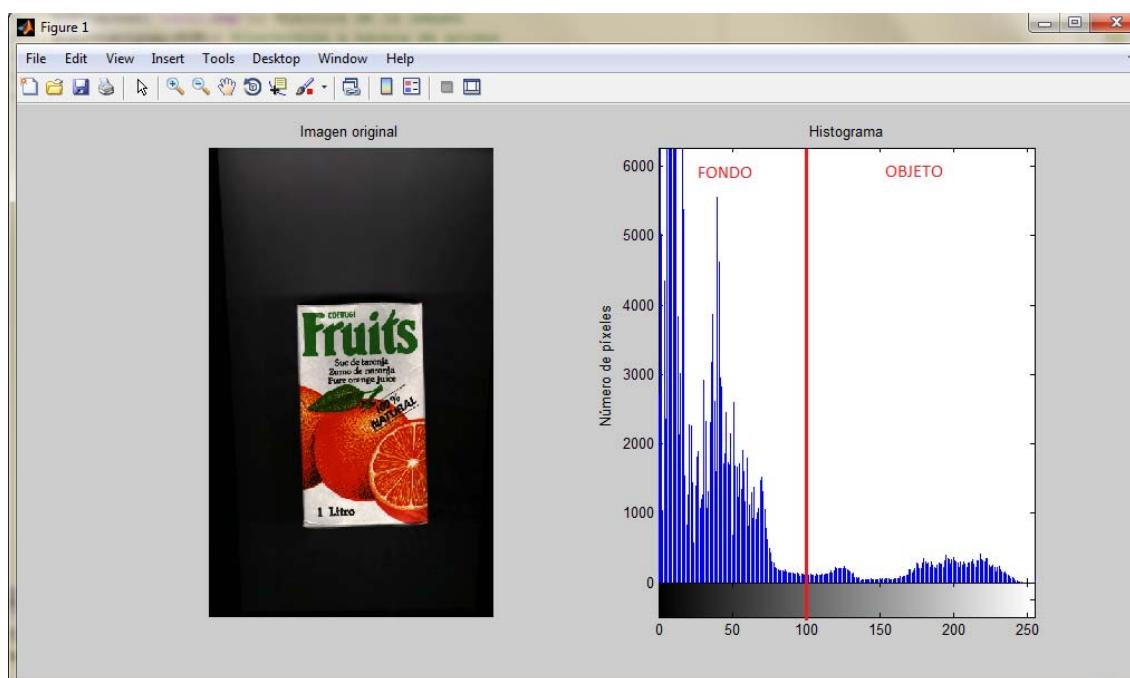
El eje horizontal corresponde a los valores de los colores: Negro, gris oscuro, gris claro y blanco, o lo que es lo mismo: 00, 01, 10 y 11. La altura de cada barra del eje vertical representa el número de píxeles de la imagen que representan un nivel de gris concreto.

Podemos sacar las siguientes conclusiones con tal solo una ojeada al *histograma*:

- El gris claro es el píxel menos común.
- El color blanco es el píxel más común.

Dado un histograma de una imagen cuyo color de fondo es diferente a la tonalidad del objeto o figura a identificar, podemos separar el fondo y la figura en dos regiones dominantes. Vamos a realizar un estudio mediante histogramas de una imagen de un tetrabrik en un fondo negro. La idea básica consiste en lograr diferenciar el fondo negro de la imagen que queremos estudiar.

En la figura 14 realizamos un histograma en la escala de grises, y como se puede apreciar, las componentes más oscuras, es decir, los valores más bajos, pertenecen al fondo. Por lo tanto, para distinguir diferentes objetos en un fondo negro simplemente nos basta con estudiar las componentes más claras, es decir, los valores más altos.



**Figura 14: Histograma en blanco y negro de un tetrabrik en un fondo negro.**

### 4.1.2 Detección de bordes.

La detección de bordes es una técnica muy clásica en el procesado digital de imagen y en la visión artificial, muy utilizada para la identificación de figuras.

Trata de buscar el contraste entre dos niveles de gris, es decir, busca los bordes mediante transiciones en la escala de grises. La técnica de detección de bordes suministra una valiosa información sobre los límites de los objetos y puede ser utilizada para segmentar y reconocer objetos.

Para la detección de bordes se utilizan diferentes aproximaciones discretas de la primera derivada ("gradiente") y de la segunda derivada ("laplaciano") de los niveles de gris de la imagen.

- Gradiente:

El gradiente de un campo escalar  $f$  es un campo vectorial, denotado como nabla  $\nabla f$ . El vector gradiente  $\nabla f$  de  $f$  para un punto genérico  $x$  del dominio de  $f$ , indica la dirección en la cual el campo  $f$  varía más rápidamente, y su módulo representa la cantidad de variación de  $f$  en la dirección de dicho vector gradiente.

Ejemplo:

Dado el campo escalar  $f$ , calculamos el gradiente  $\nabla f$ .

$$f(x, y, z) = 2x + 3y^2 - \sin(z)$$

$$\nabla f = f \left( \frac{df}{dx}, \frac{df}{dy}, \frac{df}{dz} \right) = (2,6y, -\cos(z))$$

- Laplaciano:

El laplaciano es un operador diferencial elíptico de segundo orden, denotado como delta  $\Delta$ . Expresado en coordenadas cartesianas es igual a la suma de todas las segundas derivadas parciales no mixtas dependiente de una variable.

Ejemplo:

Dado el campo escalar  $f$ , calculamos el gradiente  $\Delta f$ .

$$f(x, y, z) = 2x + 3y^2 - \sin(z)$$

$$\Delta f = \nabla \nabla f$$

$$\Delta f = 6 + \operatorname{sen}(z)$$

En conclusión, el gradiente y el operador laplaciano indican la cantidad de cambio en la escala de grises de una imagen. El operador laplaciano es muy sensible al ruido debido a que es una segunda derivada, por lo tanto, el gradiente es el método más utilizado.

#### *4.1.2.1 Operadores basados en la primera derivada:*

Las imágenes manejan un espacio vectorial bidimensional, es decir, las derivadas parciales del eje “x” e “y”. Es posible aproximar las derivadas parciales mediante diferencias entre los niveles de gris de la imagen.

$$\frac{df(x, y)}{dx} = f(x, y) - f(x - 1, y)$$

$$\frac{df(x, y)}{dy} = f(x, y) - f(x, y - 1)$$

El gradiente de la fila y de la columna en cada punto se obtienen mediante la convolución de la imagen con una máscara.

$$\nabla_{Fila}(x, y) = Imagen(x, y) \otimes H_{Fila}(x, y)$$

$$\nabla_{Columna}(x, y) = Imagen(x, y) \otimes H_{Columna}(x, y)$$

El módulo y dirección quedan definidos por el vector gradiente de la siguiente manera:

$$|\nabla(x, y)| = \sqrt{\nabla_{Fila}(x, y)^2 + \nabla_{Columna}(x, y)^2}$$

$$\nabla(x, y) = \nabla_{Fila}(x, y) + \nabla_{Columna}(x, y)$$

Una vez obtenido el valor del gradiente se decide si es un borde o no en función de un umbral prefijado. Este proceso es común a todos los operadores que explicaremos a continuación.

### Operador Roberts.

Las máscaras utilizadas son las siguientes:

$$H_{Fila}(x, y) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix}$$

$$H_{Columna}(x, y) = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

La ventaja principal es que considera muy pocos píxeles de entrada para hacer la aproximación. El principal inconveniente es su extrema sensibilidad al ruido permitiéndonos únicamente marcar los puntos de borde, es decir, su localización pero no su orientación (figura 15).



Figura 15: Imagen procesada con el operador Roberts.

El uso de pocos píxeles lo convierte en un operador muy simple, que trabaja muy bien con imágenes binarias y con una gran velocidad de cómputo.

Ventajas:

- Buena respuesta en bordes horizontales y verticales.
- Buena localización.
- Simpleza y rapidez de cálculo.

Desventajas:

- Mala respuesta en bordes diagonales.
- Sensible al ruido.
- Empleo de máscaras pequeñas.
- No da información acerca de la orientación del borde.
- Anchura del borde de varios píxeles.

### Operadores de Prewitt y Sobel.

Las máscaras usadas son las siguientes:

$$H_{Fila}(x, y) = \frac{1}{2+k} \begin{pmatrix} 1 & 0 & -1 \\ k & 0 & -k \\ 1 & 0 & -1 \end{pmatrix}$$

$$H_{Columnas}(x, y) = \frac{1}{2+k} \begin{pmatrix} -1 & -K & -1 \\ 0 & 0 & 0 \\ 1 & K & 1 \end{pmatrix}$$

#### A) Operador Prewitt, $K = 1$ .

El detector *Prewitt* no enfatiza en los píxeles cercanos al centro de la máscara al utilizar un coeficiente 1. Supervisa los niveles de gris de filas y columnas adyacentes al centro para conseguir una mayor inmunidad al ruido (figura 16).

La detección de bordes se produce de manera vertical y horizontal.

Ventajas:

- Buena respuesta en bordes horizontales y verticales.
- Poco sensible al ruido.
- Proporciona la magnitud y dirección del borde.

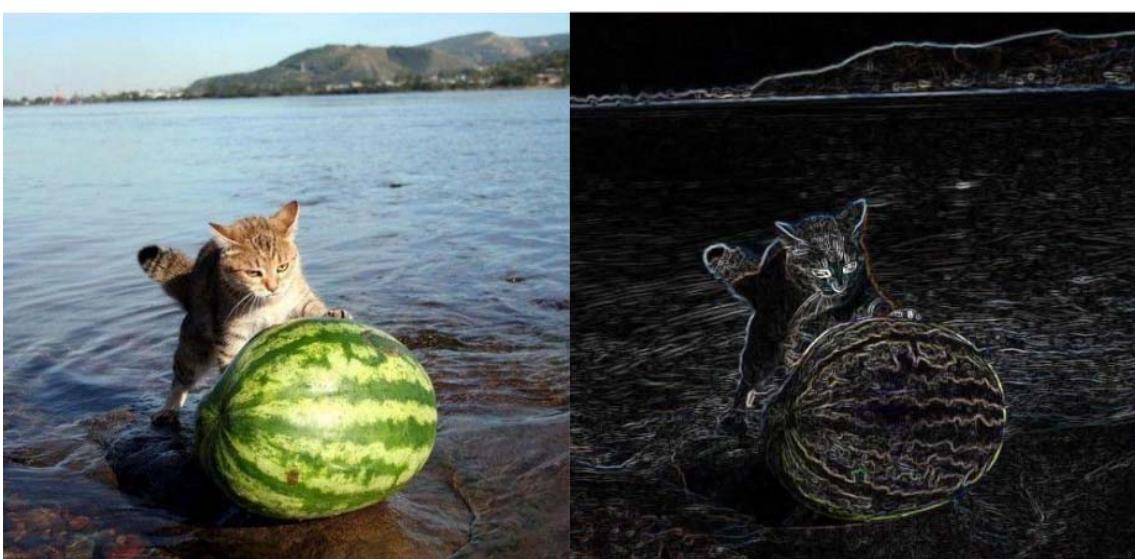


Figura 16: Imagen procesada con el operador Prewitt.

Desventajas:

- Mala respuesta en bordes diagonales.
- Lentitud de cálculo.
- No da información acerca de la orientación del borde.
- Anchura del borde de varios píxeles.

B) Operador Sobel,  $K = 2$ .

Los operadores de gradiente suelen aumentar la presencia de ruido en la imagen, no obstante, el detector *Sobel* es una combinación de un filtro de suavizado de ruido con un operador de aproximación impreciso.



Figura 17: Imagen procesada con el operador Sobel.

Sobel enfatiza con los píxeles cercanos al centro al utilizar un coeficiente de 2. Sobel es el operador más usado y en la práctica, proporciona una buena detección de bordes diagonales (figura 17).

Ventaja:

- Buena respuesta en bordes horizontales y verticales.
- Diversidad de tamaños en las máscaras.
- Proporcionan un suavizado además del efecto de derivación.

Desventajas:

- Mala respuesta en bordes diagonales.
- Lentitud de cálculo.
- No da información acerca de la orientación del borde.
- Anchura del borde de varios píxeles

### Operador Frei-Chen.

Este operador, a diferencia de los demás, permite distinguir si un borde forma parte de un contorno (línea) o simplemente es un punto aislado. El objetivo del operador *Frei-Chen* es solucionar este problema, aportando múltiples operadores simultáneamente a cada píxel y combinando los resultados.

El operador de *Frei-Chen*, es un conjunto de nueve máscaras que se pueden dividir en tres subespacios: el de bordes, el de líneas y el medio.

El subespacio de bordes y el de líneas, como es obvio, detectan los bordes y las líneas respectivamente, mientras que el subespacio medio suaviza la foto detectando zonas de intensidad uniformes (figura 18).

Subespacio de bordes:

$$H_{F1}(x, y) = \frac{1}{2\sqrt{2}} \begin{pmatrix} 1 & \sqrt{2} & 1 \\ 0 & 0 & 0 \\ -1 & -\sqrt{2} & -1 \end{pmatrix}$$

$$H_{C1}(x, y) = \frac{1}{2\sqrt{2}} \begin{pmatrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{pmatrix}$$

$$H_{F2}(x, y) = \frac{1}{2\sqrt{2}} \begin{pmatrix} 0 & -1 & \sqrt{2} \\ 1 & 0 & -1 \\ -\sqrt{2} & 1 & 0 \end{pmatrix}$$

$$H_{C2}(x, y) = \frac{1}{2\sqrt{2}} \begin{pmatrix} \sqrt{2} & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & -\sqrt{2} \end{pmatrix}$$

Subespacio de líneas:

$$H_{F3}(x, y) = \frac{1}{2} \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$H_{C3}(x, y) = \frac{1}{2} \begin{pmatrix} -1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \end{pmatrix}$$

$$H_{F4}(x, y) = \frac{1}{6} \begin{pmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{pmatrix}$$

$$H_{C4}(x, y) = \frac{1}{6} \begin{pmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{pmatrix}$$

Uniforme:

$$H_5(x, y) = \frac{1}{6} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$



Figura 18: Imagen procesada con el operador Frei-Chen.

En resumen, el operador *Frei-Chen*, está situado entre el operador *Prewitt* y *Sobel*. Este equilibrio permite detectar bordes verticales, horizontales y diagonales, además de distinguir si un punto pertenece a una línea o es independiente.

#### 4.1.2.2 Operadores basados en la segunda derivada:

Como hemos comentado en la introducción de la detección de bordes, la segunda derivada permite una localización mucho más precisa de los puntos de bordes.

Los detectores de bordes que utilizan la primera derivada dependen de un valor umbral para determinar la existencia de un punto de borde. Este valor umbral debe ser el máximo en el gradiente de intensidad de la imagen. Para precisar un poco más, utilizamos una derivada segunda donde el máximo de la primera derivada se localiza en el cruce por el cero de la segunda derivada (ver figura 19).

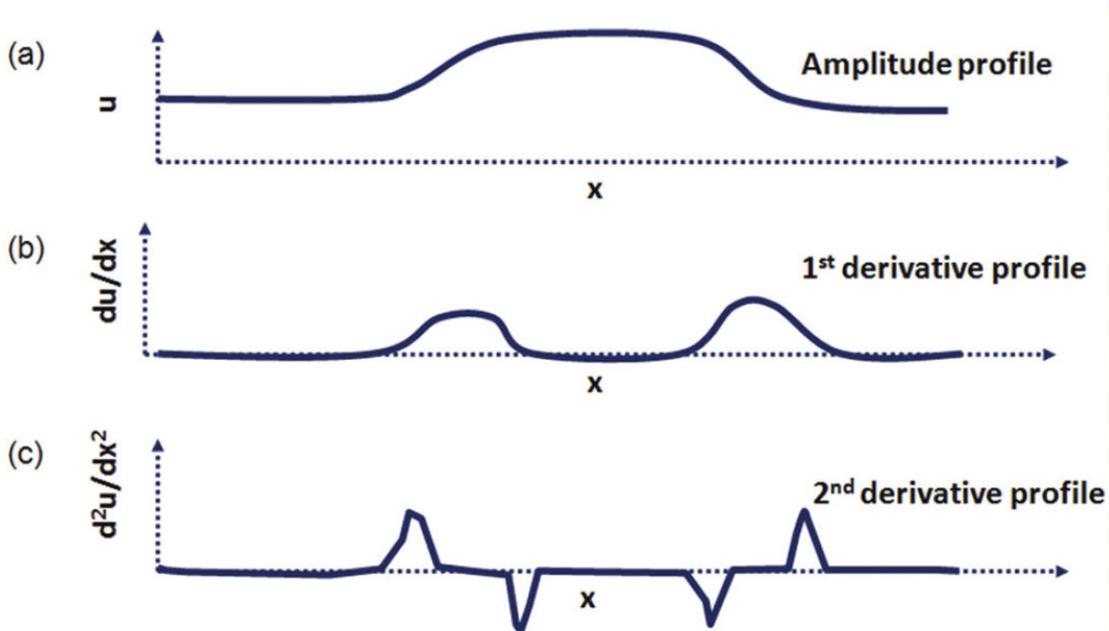


Figura 19: Representación gráfica de las diferencias entre la primera y la segunda derivada.

El operador laplaciano localiza con muchísima precisión bordes siempre que las aristas se encuentren bien separadas y la presencia del ruido en la imagen sea baja. Insistimos en que la segunda derivada es extremadamente sensible al ruido pudiéndole dar “falsos” bordes. Por lo tanto, es extremadamente recomendable suavizar previamente la imagen a procesar.

A diferencia de las máscaras de las operaciones de primera derivada, las máscaras utilizadas en este caso son simétricas rotacionalmente, permitiendo detectar bordes en todas las direcciones posibles (bordes verticales, horizontales y diagonales). Como los máximos son detectados cuando la función de la segunda derivada cruza

por cero, aplicando el teorema de Bolzano, comprobamos que existe un máximo cuando se produce un cambio de signo.

Finalmente, igual que en las operaciones de primera derivada, seleccionamos primero la máscara y realizamos una operación de convolución sobre la imagen. El signo del resultado proporciona información sobre qué lado del borde es más o menos oscuro.

### Operador Canny.

El filtro *Canny* es un algoritmo de múltiples etapas para detectar una amplia gama de bordes en imágenes [7].

El propósito de este filtro era descubrir el algoritmo óptimo de detección de bordes. Para que un detector de bordes pueda ser considerado óptimo debe cumplir los siguientes puntos:

- Buena detección: El algoritmo debe marcar el mayor número real en los bordes de la imagen como sea posible.
- Buena localización: Los bordes de marca deben estar lo más cerca posible del borde de la imagen real.
- Respuesta mínima: El borde de una imagen sólo debe ser marcado una vez, y siempre que sea posible, el ruido de la imagen no debe crear falsos bordes.

Para satisfacer estos requisitos *Canny* utiliza el cálculo de variaciones, una técnica que encuentra la función óptima para la detección de bordes. Las fases para obtener la función óptima para la detección de bordes son:

- Reducción del ruido: Se utiliza un filtro basada en la primera derivada gaussiana. Como es muy sensible al ruido presente en la imagen sin previo preprocessado, la imagen es suavizada mediante un filtro gaussiano. El resultado obtenido es una imagen un poco borrosa respecto a la original. La finalidad de este proceso es evitar que el ruido afecte individualmente a cada píxel en un grado significativo.

$$H(x, y) = \frac{1}{99} \begin{pmatrix} 9 & 12 & 9 \\ 12 & 15 & 12 \\ 9 & 12 & 9 \end{pmatrix}$$

La imagen resultante será:

$$B(x, y) = H(x, y) \otimes A(x, y)$$

- Encontrar la intensidad del gradiente de la imagen: El borde de la imagen puede apuntar en diferentes direcciones, por lo tanto el algoritmo Canny utiliza

cuatro filtros para detectar horizontal, vertical y diagonalmente los bordes de la imagen borrosa. Por ejemplo, el operador Roberts, Prewitt y Sobel, estudiados anteriormente. El uso de estos operadores devuelve un valor para la primera derivada en la dirección horizontal y la dirección vertical, a partir de este se puede calcular gradiente de borde y dirección óptima:

$$\text{Gradiente} = \sqrt{\text{Gradiente}_x^2 + \text{Gradiente}_y^2}$$

$$\phi = \arctan\left(\frac{\text{Gradiente}_y}{\text{Gradiente}_x}\right)$$

## 4.2 Conclusión sobre el modelo de procesado de imagen a utilizar en el proyecto.

Tras el estudio de las técnicas de procesado digital de imagen basadas en histogramas y detección de bordes, concluimos que la solución más viable para nuestra aplicación es el uso de histogramas. Partiendo de que el objetivo del proyecto es la implementación hardware de alguna solución que permita comparar imágenes y establecer un grado de similitud entre ellas, lo más razonable es escoger un algoritmo eficaz y sencillo.

La detección de bordes, mediante operadores de la primera y la segunda derivada, es una solución muy eficiente. La complejidad de la solución es razonable si el proyecto se implementara mediante herramientas software de procesado digital de imagen como *Matlab* u *OpenCV*.

Dado que parto de muy poca experiencia en el diseño de sistemas hardware complejos, de que este proyecto ha supuesto mi iniciación en el procesado digital de imágenes, y de que el tiempo para la realización del TFG está acotado, el desarrollo de una aplicación hardware basada en operadores de primera o segunda derivada podría resultar muy complicado. La identificación mediante histogramas quizás no es la mejor solución para el problema planteado, pero sí la solución más viable y realista para abordar con éxito este TFG.

Suponiendo un escenario y unos objetos seleccionados previamente, el uso de histogramas es suficientemente eficaz y fiable para la implementación de la aplicación.

## 11. Distancias entre la imagen de referencia y la imagen muestreada.

En este último apartado abordaremos los conceptos asociados a la “comparación” de histogramas. La comparación de histogramas es un método de procesado de imagen que permite la detección de objetos y figuras mediante el cómputo de la similitud entre los histogramas.

Para comparar un histograma de referencia ( $H_1$ ) con otro histograma ( $H_2$ ), primero hay que elegir una métrica adecuada  $d(H_1, H_2)$  que exprese la distancia (grado de similitud) entre ellos. Básicamente, todas las métricas de distancia se basan en ir calculando y almacenando el error relativo de cada uno de los términos que conforman el histograma. Por lo tanto, que un histograma sea más parecido a uno u otro, simplemente es que posea un mayor o menor grado de diferencia entre las componentes que forman cada uno de los histogramas.

En conclusión, la distancia entre dos histogramas se puede interpretar como el sumatorio de los errores obtenidos al comparar cada uno de los componentes del histograma de la imagen muestreada ( $H_2$ ) respecto al histograma de la imagen de referencia ( $H_1$ ).

## a. Métricas utilizadas para el cálculo de distancias.

Los principales métodos para el cálculo de distancia entre histogramas son los que vamos a explicar en los siguientes apartados. En cada una de las expresiones matemáticas desarrolladas en este apartado teórico,  $H_1$  corresponde con el histograma de referencia mientras que  $H_2$  corresponde con el histograma de muestreo.

### i. Correlación.

La correlación expresa el grado de proporcionalidad entre dos variables estadísticas. Se considera que dos variables estadísticas, A y B, están correlacionadas cuando los valores de una de ellas varían sistemáticamente respecto a los valores homónimos de la otra, dicho de otra manera si aumenta A también lo hará B.

En el procesado de imagen se utilizan coeficientes de correlación porque aportan una estadística muy descriptiva que se utiliza para realizar mediciones cuantitativas, donde las variables estadísticas se organizan en grupos. La correlación describe con gran exactitud la dependencia entre las unidades que conforman cada uno los grupos a analizar. Por lo tanto, en el caso de los histogramas, describe con gran exactitud la dependencia entre los elementos que conforman un histograma, extrayendo de manera muy eficaz la información que tienen en común los histogramas a comparar.

$$d(H_1, H_2) = \frac{\sum_i (H_1(i) - \bar{H}_1)(H_2(i) - \bar{H}_2)}{\sqrt{\sum_i (H_1(i) - \bar{H}_1)^2 (H_2(i) - \bar{H}_2)^2}}$$

$$\bar{H}_k = \frac{1}{N} \sum_j H_k(j)$$

## ii. Prueba $\chi^2$

La prueba  $\chi^2$  de Pearson, también llamada *Chi-cuadrado* [8], es una prueba no paramétrica que mide la discrepancia entre una distribución observada y una de referencia. Se considera no paramétrica porque su distribución no puede ser definida a priori, pues son los datos observados los que la determinan. Finalmente, indica qué grado de diferencia existe entre la distribución observada y la referencia, mediante el contraste de ambas.

$$d(H_1, H_2) = \sum_i \frac{(H_2(i) - H_1(i))^2}{H_1(i) + H_2(i)}$$

## iii. Intersección.

La intersección de dos distribuciones es una operación que resulta en otra distribución que contiene los elementos comunes a las distribuciones de partida.

### Ejemplo en teoría de conjuntos:

La intersección de dos conjuntos  $A$  y  $B$  es otro conjunto  $A \cap B$  cuyos elementos son los elementos comunes a  $A$  y  $B$ :

$$A = \{2, 4, 6, 8, 10\}$$

$$B = \{1, 4, 9, 11, 15\}$$

$$A \cap B = \{4\}$$

Aplicando esta teoría a nuestro proyecto, debemos ir recorriendo cada uno de los histogramas y generar un nuevo histograma con la cantidad de elementos en común.

$$d(H_1, H_2) = \sum_i \text{Mínimo}(H_1(i), H_2(i)) \text{ si } H_1(i) \neq H_2(i)$$

Aplicando la ecuación anterior calculamos el grado de aproximación a un conjunto disjunto.

## iv. Método de Bhattacharyya.

La distancia Bhattacharyya se utiliza con éxito en la ingeniería y las ciencias estadísticas [9].

$$d(H_1, H_2) = \sqrt{1 - \frac{1}{\bar{H}_1 \bar{H}_2 N^2} \sum_i \sqrt{H_1(i) - H_2(i)}}$$

El resultado del método de **Bhattacharyya** refleja el grado de no-similitud entre dos distribuciones de probabilidad, su principio de funcionamiento radica en la propiedad de divergencia de una distribución u otra.

### b. Conclusión sobre la métrica a utilizar para el cálculo de distancias.

Después de estudiar las diferentes alternativas para comparar un histograma de referencia con los histogramas de las distintas imágenes muestreadas, proponemos implementar en este proyecto la prueba  $\chi^2$  de Pearson, también llamada *Chi-cuadrado*.

Las principales razones para la elección de esta métrica son las siguientes:

1. El valor de la distancia Chi-cuadrada nunca es negativa, ya que el valor entre las dos frecuencias es elevada al cuadrado.
2. Es una métrica no paramétrica, es decir, es factible de ser utilizada con muestras pequeñas. No se necesita suposiciones restrictivas de las pruebas paramétricas y se utilizan con datos cualitativos siendo fácil de comprender.
3. Es mucho más sencilla de implementar en VHDL que los métodos de correlación o **Bhattacharyya**.

## 12. Conclusión sobre la arquitectura de sistema propuesta en el proyecto.

El resultado esperado en este proyecto es la implementación de un sistema hardware capaz de almacenar en una memoria una escena de referencia y calcular el grado de similitud con las siguientes escenas fotografiadas.

Después del estudio teórico realizado y teniendo en cuenta las conclusiones de los apartados 3.5, 4.2 y 5.2 de este anexo, se propone la implementación de un sistema que realice las siguientes funciones:

- 1.- Adquirir una imagen de modelo RGB mediante el módulo *VmodCAM* de *Digilent*.
- 2.- Pre-procesar la imagen RGB para obtener otra de modelo HSL.
- 3.- Calcular los histogramas de las componentes H (tono), S (saturación) y L (luminosidad) de la nueva imagen (obtener los histogramas  $H_2$ ,  $S_2$  y  $L_2$ ).

- 4.- Utilizando la prueba  $\chi^2$  de Pearson calcular el grado de similitud (comparar) de los histogramas  $H_2$ ,  $S_2$  y  $L_2$  con los de la imagen de referencia ( $H_1$ ,  $S_1$  y  $L_1$ ), que habrán sido previamente calculados y almacenados en una memoria.



## 13.

# ANEXO 4: Prototipo mediante Matlab

## Contenido.

1.	Introducción .....	66
1.1	¿Cómo trabaja <i>Matlab</i> con imágenes? .....	66
1.2	Tipos de datos en <i>Matlab</i> . ....	66
1.3	Tipos de imágenes en <i>Matlab</i> . ....	66
1.3.1	Imágenes indexadas:.....	66
1.3.2	Imágenes en escala de grises: .....	67
1.3.3	Imágenes binarias:.....	67
1.3.4	Imágenes en RGB:.....	68
1.4	Funciones básicas para el procesado de imágenes en <i>Matlab</i> . ....	69
1.4.1	Leer y escribir una imagen:.....	69
1.4.2	Visualización de imágenes:.....	69
1.4.3	Conversión del tipo de datos.....	70
1.4.4	Conversión del tipo de imagen.....	70
2	Colorimetría .....	71
2.1	Imágenes RGB.....	71
2.2	Imágenes en la escala de grises.....	73
2.3	Algoritmo para la conversión imágenes RGB a HSL .....	74
2.3.1	Módulo Lightness:.....	77
2.3.2	Módulo Saturation.....	77
2.3.3	Módulo Hue.....	77
2.3.4	Código y ejemplos.....	78
3	Histogramas .....	80

3.1	Histogramas en blanco y negro.....	80
3.1.1	Histogramas formados por colores oscuros.....	80
3.1.2	Histogramas formados por colores claros.....	81
3.1.3	Histograma estrecho.....	82
3.1.4	Histograma con gran contraste.....	83
3.2	Histogramas a color. ....	84
3.3	Algoritmo para la creación de un histograma HSL.....	85
4	Cálculo de distancias entre histogramas.....	88

## 1. Introducción.

En este anexo se realiza un estudio que conduce al diseño en *Matlab* de un prototipo del sistema de procesado de imagen propuesto en el proyecto. Con ello pretendemos llevar a la práctica los conceptos teóricos analizados en el anexo anterior. Así mismo se pretende disponer de un modelo del sistema a implementar en la FPGA para garantizar el correcto funcionamiento de los algoritmos de procesado de imagen [4][10].

*Matlab* (abreviatura de *MAT*rix *LAB*oratory) es una excelente herramienta de software matemático que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio. Dispone de un conjunto de funciones específicas de procesado de imagen.

### 1.1 ¿Cómo trabaja *Matlab* con imágenes?

La estructura más básica de datos en *Matlab* es el vector, un conjunto de datos ordenados, que permite la realización de matrices. Las imágenes en *Matlab* son representadas como matrices cuyos valores indican colores e intensidades. Las filas y las columnas localizan cada uno de los píxeles de una imagen digital configurando la imagen como una matriz matemática. Por ejemplo, una imagen 640x480 formará una matriz de 640 filas y 480 columnas.

### 1.2 Tipos de datos en *Matlab*.

*Matlab* utiliza los siguientes tipos de datos para implementar un vector:

- Double: Los datos de estos vectores son números de 64 bits en coma flotante.
- Unit8: Los datos de estos vectores son números de 8 bits positivos sin signo, es decir, del 0 al 255.
- Unit16: Igual que el *unit8* pero con mayor capacidad, los datos de estos vectores son números de 16 bits positivos sin signo, es decir, del 0 al 65535.

### 1.3 Tipos de imágenes en *Matlab*.

Existen cuatro tipos básicos de imágenes en *Matlab*:

#### 1.3.1 Imágenes indexadas:

Una imagen indexada consiste en una matriz-imagen y un mapa de color. Los píxeles de la matriz-imagen tienen valores que son índices (apuntan) a un mapa de color. Por lo tanto, la matriz-imagen forma la imagen a través de una paleta de colores.

La matriz-imagen es representada por un vector del tipo *unit8*, *unit16* o *double*, mientras que el mapa de color es siempre un vector de *mx3* del tipo *double*.

7	1	7
1	7	1
1	7	1

Número	Color RGB mx3
1	Rojo
2	Naranja
3	Amarillo
4	Verde
5	Azul
6	Añil
7	Violeta

**Figura 1. Ejemplo de una imagen indexada.**

Por ejemplo, el píxel con valor de dato 1 apunta a la primera columna del mapa de color, el píxel con valor de dato 2 apunta a la segunda columna del mapa de color, el píxel con valor de dato 3 apunta a la tercera columna del mapa de color... y así sucesivamente.

### 1.3.2 Imágenes en escala de grises:

Una imagen en escala de grises es representada mediante una matriz cuyos píxeles indican las intensidades de gris. Los datos de la matriz pueden ser *double*, *uint16* o *uint8*, permitiéndonos un mayor o un menor rango de grises.

Como he explicado en el anexo anterior, el valor 0 representa el valor de intensidad nula, es decir, el color negro, mientras que el valor máximo representa el color blanco.

### 1.3.3 Imágenes binarias:

Una imagen binaria es representada mediante una matriz cuyos píxeles pueden tener únicamente dos valores, '0' (negro) o '1' (blanco). Los datos de la matriz pueden ser *double*, *uint16* o *uint8*, siendo, como es obvio, preferible el uso de datos tipo *uint8*. La imagen binaria sólo utiliza 2 valores, por lo tanto, es independiente del rango de valores del tipo de dato utilizado, siendo más eficiente en memoria el de menor rango de representación.

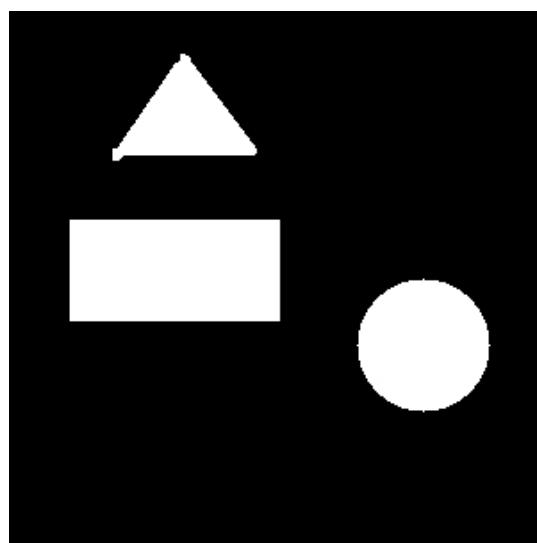


Figura 2. Ejemplo de una imagen binaria.

#### 1.3.4 Imágenes en RGB:

Una imagen RGB es representada en *Matlab* como un vector  $mxnx3$  que define los componentes rojo, verde y azul de cada píxel. Como he dicho en el anexo anterior, el color de cada uno de los píxeles se obtiene por la combinación de los componentes rojos, verdes y azules, por ese motivo, las imágenes RGB están formadas por 3 matrices  $mxn$ . Por ejemplo, el píxel cuya posición es  $matriz(i,j)$  está formado por: componente roja  $matriz(i,j,1)$ , componente verde  $matriz(i,j,2)$  y componente azul  $matriz(i,j,3)$ .



Figura 3. Ejemplo de una imagen RGB.

El tipo de datos utilizado para representar cada uno de los componentes que forman el color son *double*, *uint8* y *uint16*:

- Uint8:

$$\text{rango de colores} = 2^{8+8+8}$$

- Uint16:

$$\text{rango de colores} = 2^{16+16+16}$$

- Double:

$$\text{rango de colores} = 2^{64+64+64}$$

## 1.4 Funciones básicas para el procesado de imágenes en Matlab.

En este apartado voy a citar las funciones más habituales de procesado de imagen con *Matlab*.

### 1.4.1 Leer y escribir una imagen:

La función *imread* lee una imagen de un fichero con un formato soportado por *Matlab* y la almacena en memoria como una matriz RGB del tipo *unit8*.

```
imagenRGB = imread('fichero.jpg');
```

La función *imwrite* crea un fichero con uno de los formatos soportados por *Matlab* para guardar una imagen almacenada en una variable del *workspace*.

Para imágenes de tipo binario, escala de grises y RGB:

```
imwrite(imagenRGB, 'fichero.jpg');
```

Para imágenes indexadas:

```
imwrite(X, map, 'fichero.jpg');
```

### 1.4.2 Visualización de imágenes:

Mediante la función *imshow* podemos visualizar imágenes almacenadas en variables del *workspace*.

Para imágenes de tipo binario, escala de grises y RGB:

```
imshow(imagen);
```

Para imágenes indexadas:

```
imshow(X, map);
```

### 1.4.3 Conversión del tipo de datos

Para realizar funciones para el procesado digital de imágenes, es muy importante tener en cuenta que no podemos hacer operaciones aritméticas sobre vectores del tipo *uint8* y *unit16*. Por lo tanto, lo primero que debemos hacer es convertir la imagen original a vectores del tipo *double*, realizar el procesado oportuno y finalmente convertir la imagen al formato *uint8*.

#### Conversión de uint8 a double:

- Indexada:

$$B = \text{double}(A) + 1;$$

- Escala de grises o RGB:

$$B = \text{double}(A)/255;$$

- Binaria

$$B = \text{double}(A);$$

También podemos utilizar el método *im2double*.

#### Conversión de double a uint8:

- Indexada:

$$B = \text{uint8}(\text{round}(A - 1));$$

- Escala de grises o RGB:

$$B = \text{uint8}(\text{round}(A * 255));$$

- Binaria

$$B = \text{logical}(\text{unit8}(\text{round}(A)));$$

También podemos utilizar el método *im2uint8*.

### 1.4.4 Conversión del tipo de imagen

*im2bw*: Crea una imagen binaria a partir de una imagen en escala de grises.

*gray2ind*: Crea una imagen indexada a partir de una imagen en escala de grises.

*ind2gray*: Crea una imagen en la escala de grises a partir de una imagen indexada.

*ind2rgb*: Crea una imagen RGB a partir de una imagen indexada.

*rgb2gray*: Crea una imagen en la escala de grises a partir de una imagen en RGB.

*rgb2ind*: Crea una imagen indexada a partir de una imagen en RGB.

mat2gray: Crea una imagen en la escala de grises a partir de una matriz de datos numérica.

## 14. 2 Colorimetría.

En este apartado, haremos una aplicación práctica de las ideas de colorimetría presentadas en el marco teórico, culminando con la opción de representación de los colores elegida, que es, HSL (Hue, Saturation and Lightness). Presentaremos todos los pasos necesarios para la realización del algoritmo de conversión de RGB a HSL, una simulación y un prototipo de las ideas recogidas en *Matlab*.

### 2.1 Imágenes RGB.

Leemos una imagen mediante la función *imread* y posteriormente la visualizamos con la función *imshow*.

#### Código RGBimage.m

```
%RGB image
image='tcol1.bmp';
Im = imread(image);
title('Imagen RGB');
imshow(Im),title('Imagen original');
 xlabel('Largo');
 ylabel('Alto');
```

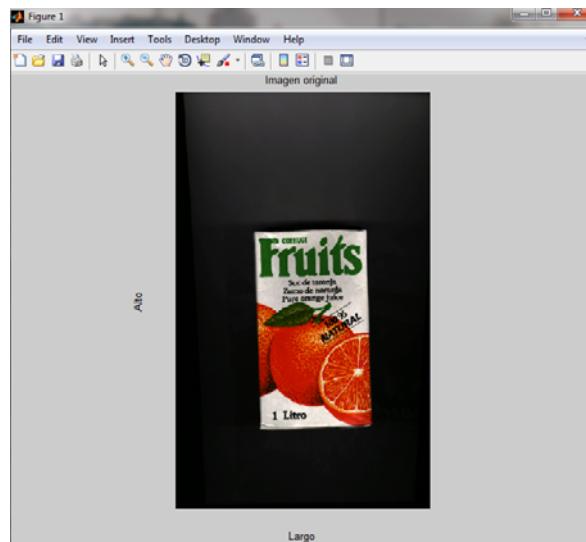


Figura 4. Imagen tcol1.bmp.

Descomponemos una imagen RGB en cada una de sus componentes de colores: Componente roja  $matriz(i,j,1)$ , componente verde  $matriz(i,j,2)$  y componente azul  $matriz(i,j,3)$ .

### Código RGBimage2.m

```
%RGB image 2
image=('Colors.jpg');
Im = imread(image);
Red=Im(:,:,1);
Green=Im(:,:,2);
Blue=Im(:,:,3);
title('Imagen RGB');
subplot(2,2,1),imshow(Im),title('Imagen original');
xlabel('Largo'),ylabel('Alto');
subplot(2,2,2),imshow(Red),title('Componente roja');
xlabel('Largo'),ylabel('Alto');
subplot(2,2,3),imshow(Green),title('Componente verde');
xlabel('Largo'),ylabel('Alto');
subplot(2,2,4),imshow(Blue),title('Componente azul');
xlabel('Largo'),ylabel('Alto');
```

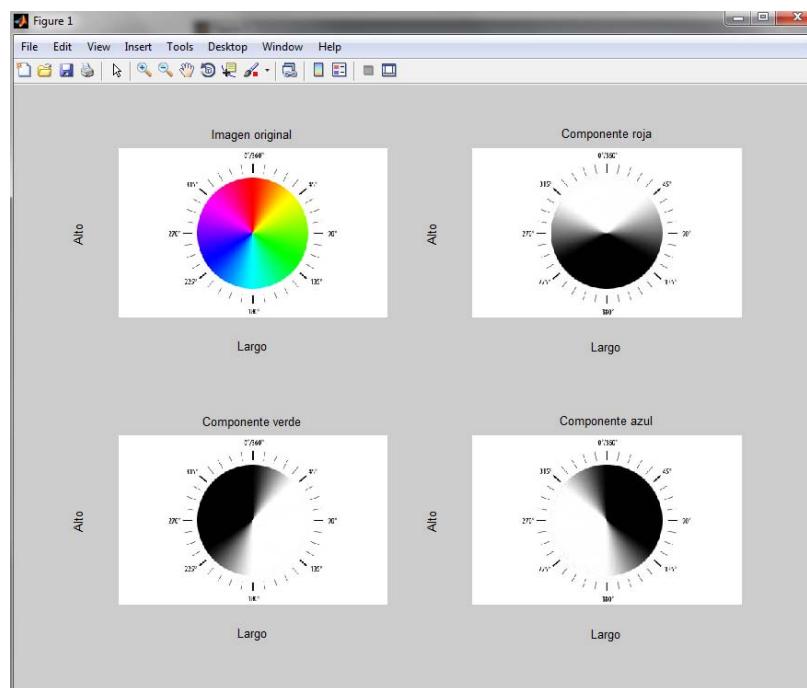


Figura 5. Resultados tras ejecutar las funciones RGBimage.m y RGBimage2.m.

## 2.2 Imágenes en la escala de grises.

Utilizamos la función *rgb2gray* para poder convertir una imagen RGB a la escala de grises.

### Código Grayimage.m

```
%Gray image
image='Lenna.png';
Im = imread(image);
Gray=rgb2gray(Im);
title('Imagen escala de grises');
subplot(1,2,1),imshow(Im),title('Imagen original');
xlabel('Largo'),ylabel('Alto');
subplot(1,2,2),imshow(Gray),title('Imagen escala de grises');
xlabel('Largo'),ylabel('Alto');
```

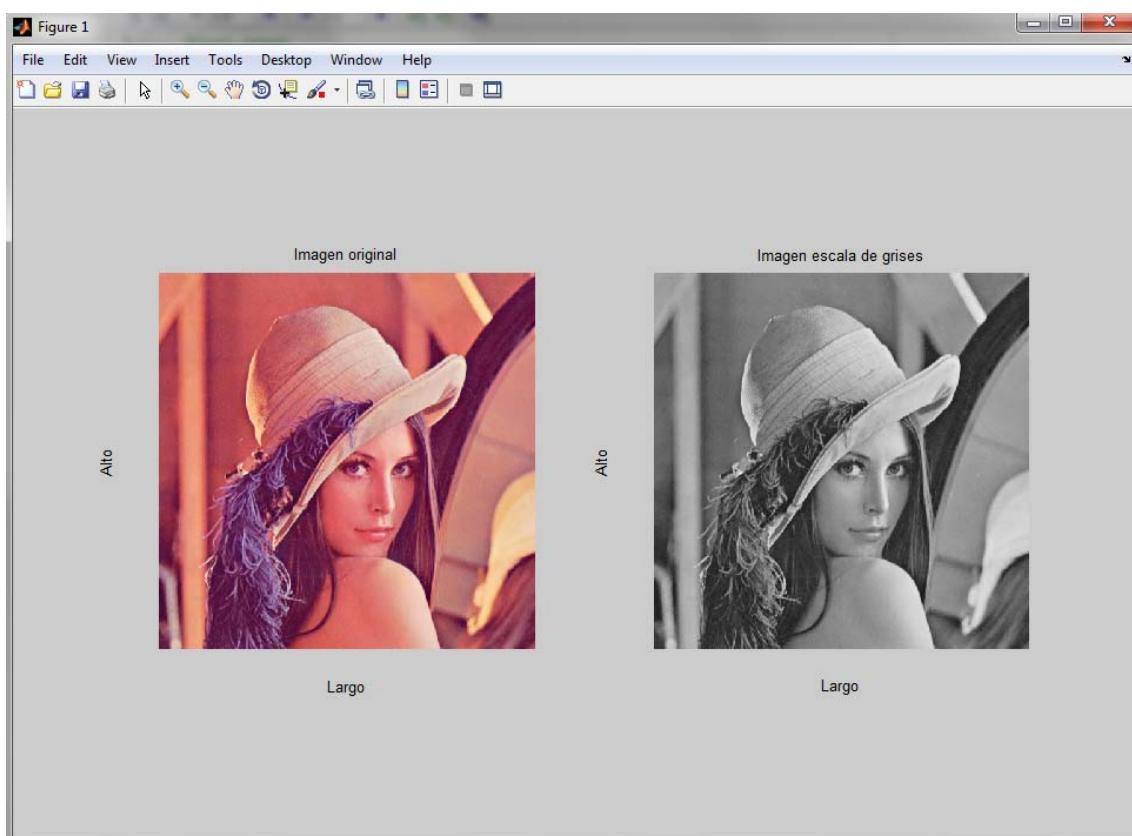


Figura 6. Resultados tras ejecutar la función Grayimage.m.

He programado una función alternativa a `rgb2gray` (incluida en las librerías de *Matlab*) que realiza la misma función. La finalidad es puramente didáctica y me permite desgranar un algoritmo práctico para la conversión de imágenes en la FPGA.

### Código `RGBtoGray.m`

```
function Gray = RGBtoGray(Im);
%Converts an RGB image to grayscale
Red=Im(:,:,1);
Green=Im(:,:,2);
Blue=Im(:,:,3);
Gray = 0.3*Red + 0.59*Green + 0.11*Blue;
```

## 2.3 Algoritmo para la conversión imágenes RGB a HSL.

Una vez completados los conceptos básicos de colorimetría en *Matlab*, procedemos con el análisis del modelo de colorimetría que necesitamos. HSL es el modelo idóneo para nuestro proyecto, ya que nos permite un procesado de imagen independiente de la componente lumínica. Por lo tanto, apoyándonos en las bases teóricas investigadas, procedemos con la realización de una simulación y un prototipo de conversión de modelos de colorimetría en *Matlab*.

La conversión RGB a HSL actúa píxel a píxel, es decir, cada píxel es convertido de un modelo a otro de manera independiente de los demás. Dicho de otra manera, para conseguir una conversión completa de RGB a HSL necesitamos realizar una conversión individual de cada uno de los píxeles que conforman la imagen.

La figura 7 muestra el diagrama de flujo del algoritmo de conversión del formato RGB al HSL.

Tenemos que ir recorriendo la matriz de valores RGB e ir recalculando los nuevos valores para la matriz HSL. Es decir, ir desplazando el eje de coordenadas ‘x’ e ‘y’ de las matrices *R*, *G* y *B* (*R*(*i,j*), *G*(*i,j*) y *B*(*i,j*) respectivamente).

$$1 < i < m \text{ filas de } R, G \text{ o } B$$

$$1 < j < n \text{ columnas de } R, G \text{ o } B$$

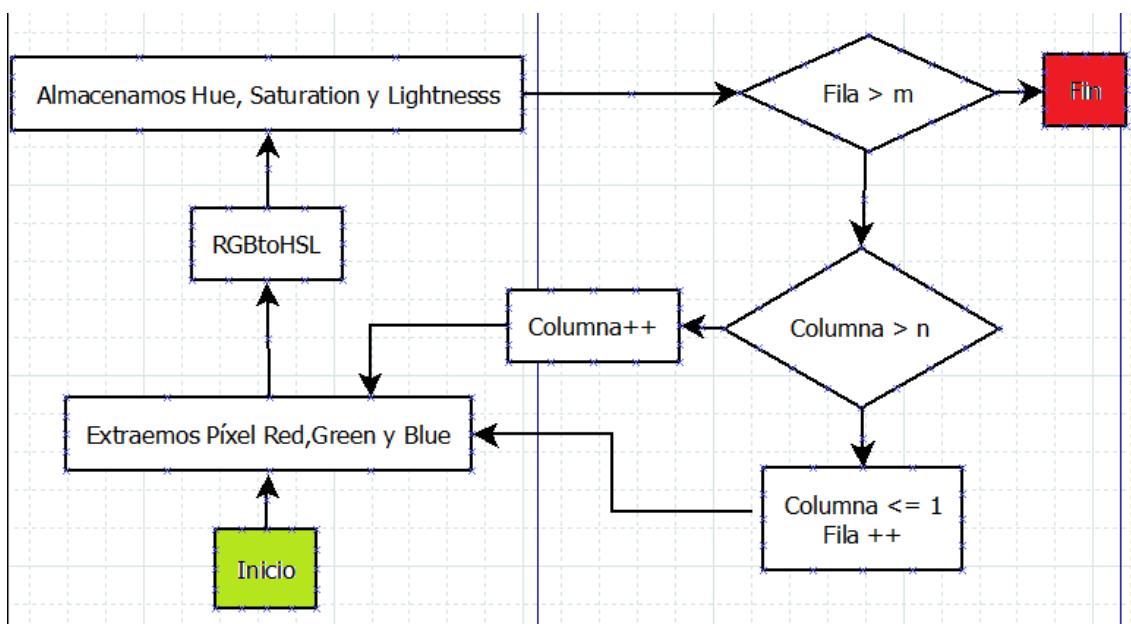


Figura 7. Diagrama de flujo del algoritmo de conversión RGB a HSL.

Para calcular los valores del modelo HS (figura 8) lo primero que tenemos que hacer es identificar el valor de menor peso y el valor de mayor peso de las componentes de color RGB, es decir, extraer el máximo y el mínimo de los valores RGB que componen el píxel. Los valores del máximo y del mínimo se utilizarán a lo largo del algoritmo de conversión.

$$M = \max (R(i,j), G(i,j), B(i,j))$$

$$m = \min (R(i,j), G(i,j), B(i,j))$$

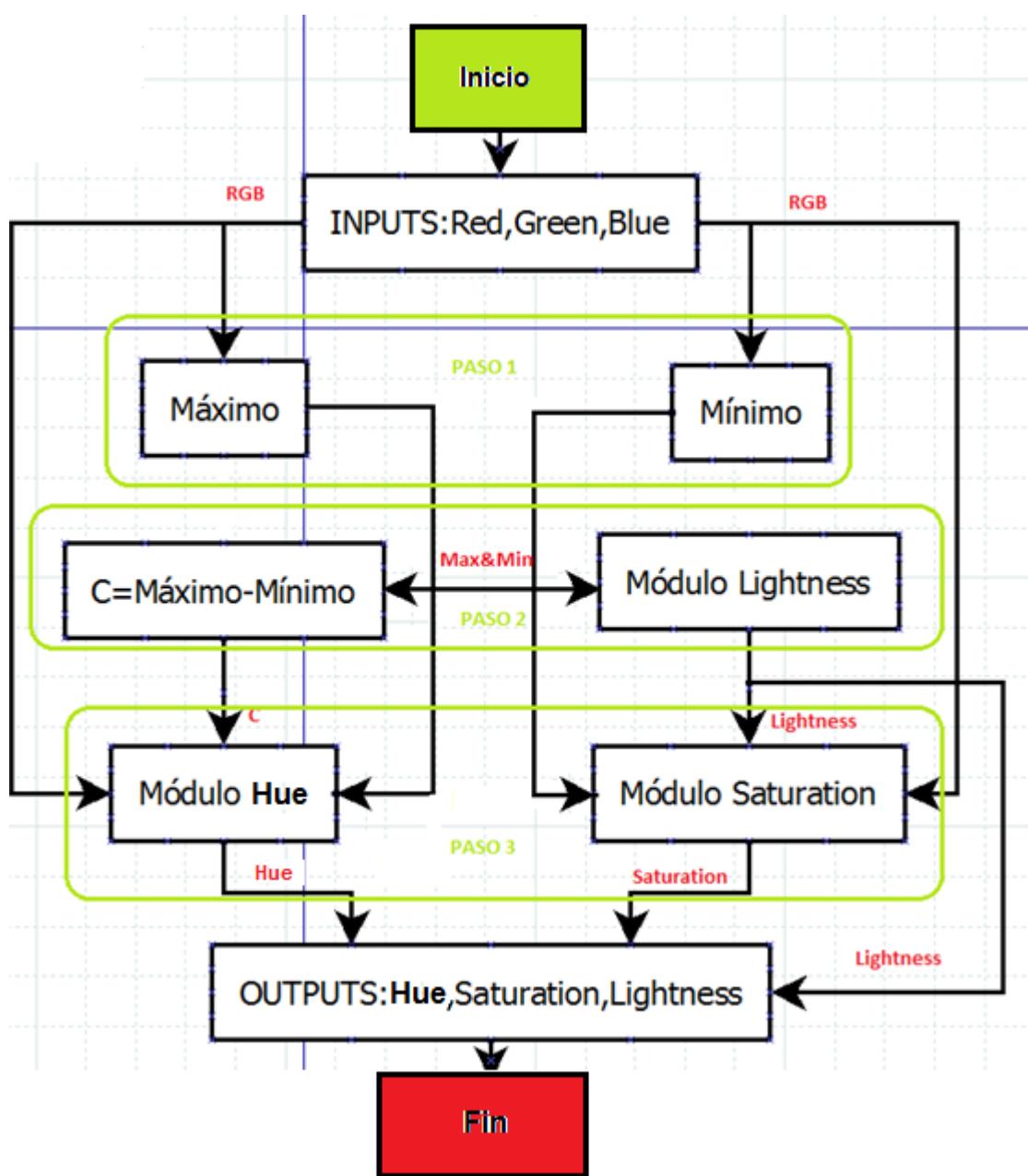


Figura 8. Diagrama de flujo de la función RGBtoHSL.

El segundo paso del algoritmo es el cálculo del parámetro C, que es la diferencia entre el valor máximo y mínimo. También calculamos la componente Lightness mediante módulo Lightness.

$$C = M - m$$

El último paso, es el cálculo de las componentes *Hue* y *Saturation* mediante el *módulo Hue* y el *módulo Saturation* respectivamente.

### 2.3.1 Módulo Lightness:

Con estas operaciones, como es obvio, calculamos la componente lumínica del píxel.

$$\text{Lightness} = 50 * (M + m)$$

Son operaciones muy sencillas, simplemente tenemos que sumar el máximo y el mínimo obtenidos en el primer paso, y posteriormente, multiplicamos dicha suma por 50.

### 2.3.2 Módulo Saturation.

Este módulo calcula la componente de saturación del píxel. Lo primero que tenemos que hacer es identificar cada uno de los casos posibles dependiendo del valor de la componente lumínica obtenida con el *módulo Lightness*.

$$\text{Saturation} = 0 \text{ si Lightness} = 0$$

$$\text{Saturation} = \frac{C}{M + m} \text{ si Lightness} < 50$$

$$\text{Saturation} = \frac{C}{2 - M - m} \text{ si Lightness} \geq 50$$

Una vez distinguido cada uno de los casos, debemos realizar la operación asignada para obtener el valor de saturación.

### 2.3.3 Módulo Hue.

Igual que en el *módulo Saturation* lo primero que tenemos que hacer es distinguir cada uno de los casos posibles, esta vez, dependiendo de los valores máximo y mínimo.

Excepto en el primer caso, en el cuál, se cumple que el máximo y el mínimo son iguales, los demás casos dependen de la localización del máximo en la componente de colores RGB. Es decir, identificamos tres casos dependiendo de si el máximo está localizado en la componente roja, verde o azul.

$$H(i, j) = 0 \text{ si } M = m$$

$$H(i,j) = \text{mod} \left( 60 * \frac{G(i,j) - B(i,j)}{C} + 360, 360 \right) \text{ si } M = R(i,j)$$

$$H(i,j) = 60 * \frac{B(i,j) - R(i,j)}{C} + 120 \text{ si } M = G(i,j)$$

$$H(i,j) = 60 * \frac{R(i,j) - G(i,j)}{C} + 240 \text{ si } M = B(i,j)$$

Finalmente realizamos las operaciones asignadas a cada uno de los respectivos casos.

#### 2.3.4 Código y ejemplos.

##### Código ejemploRGBtoHSL.m

```
imagen= ('tcol1.bmp') ;
Im = imread(imagen) ;
[H,S,L]=rgb2hsl(Im) ;
```

##### Código función rgb2hsl.m

```
function [H,S,L]=rgb2hsl(RGB)
rgb=im2double(RGB); %Pasamos de RGB en unit8 a RGB en double

R=rgb(:,:,1);
G=rgb(:,:,2);
B=rgb(:,:,3);
[Y,X]=size(R);
H=zeros(Y,X); %Tamaño X*Y
S=zeros(Y,X);
L=zeros(Y,X);

for i=1:Y
    for j=1:X
        [H(i,j),S(i,j),L(i,j)]=methodrgb2hsl(R(i,j),G(i,j),B(i,j));
    end
end
end
```

##### Código función methodrgb2hsl.m

```
function [H,S,L]=methodrgb2hsl(R,G,B)
vector=[R,G,B];
M=max(vector);
m=min(vector);
C=(M-m);

L=LightnessModule(M,m);
S=SaturationModule(M,m,L,C);
H=HueModule(M,m,R,G,B);
end
```

### Código función LightnessModule.m

```
function L=LightnessModule(M,m)
Laux=50*(M+m);
L=redondeocaballero(Laux);
end
```

### Código función SaturationModule.m

```
function S=SaturationModule(M,m,L,C)
if M==m
    Saux=0;
elseif (L) < 50
    Saux=(C/(M+m))*100;
else
    Saux=(C/(2-M-m))*100;
end
S=redondeocaballero(Saux);
end
```

### Código función HueModule.m

```
function H=HueModule(M,m,C,R,G,B)
if M==m
    Haux=0;
elseif M==R
    Haux=mod(60*((G-B)/C)+360,360);
elseif M==G
    Haux=60*((B-R)/C)+120;
else
    Haux=60*((R-G)/C)+240;
end
H=redondeocaballero(Haux);
end
```

Añado una función extra llamada *redondeocaballero* que sirve para redondear resultados obteniendo únicamente números enteros.

### Código función redondeocaballero.m

```
function R=redondeocaballero(N)
decimal=N-fix(N);
if N>=0
    if decimal>=0.5
        R=fix(N)+1;
    else
        R=fix(N);
    end
else
    if decimal<=-0.5 R=fix(N)-1;
    else
        R=fix(N);
    end
end
```

## 15. 3 Histogramas

### 3.1 Histogramas en blanco y negro.

Para poder ver el histograma de una imagen basta con utilizar la función *imhist*. La función *imhist* opera con imágenes en escala de grises, por lo que previamente, convertiremos la imagen *RGB* a escala de grises.

#### 3.1.1 Histogramas formados por colores oscuros.

Como podemos observar en el histograma del ejemplo (figura 9), los colores más predominantes de la foto son los colores oscuros. La foto de la catedral de noche, como es obvio, al ser una foto nocturna, carece prácticamente de matrices brillantes.

#### Código ejemploHistogramCathedral.m

```
%Histogram Cathedral
image= ('Cathedral.jpg');
Im1 = imread(image);
Im=RGBtoGray(Im1);
title('Histograma Cathedral');
subplot(2,2,1),imshow(Im1),title('Imagen original
RGB'), xlabel('Largo'), ylabel('Alto');
subplot(2,2,2),imshow(Im),title('Imagen original Escala de
grises'), xlabel('Largo'), ylabel('Alto');
subplot(2,2,3),imhist(Im),title('Histograma'), xlabel('Rango de
grises'), ylabel('Número de píxeles');
```

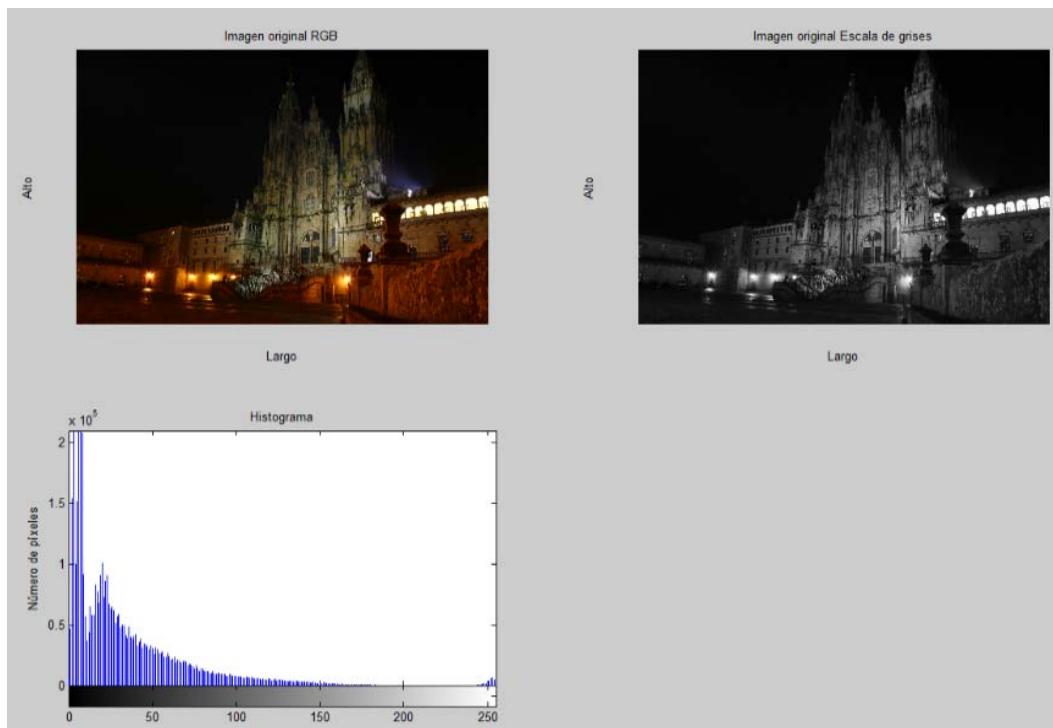


Figura 9. Histograma formado por colores oscuros.

### 3.1.2 Histogramas formados por colores claros.

Este es el caso contrario al anterior, los colores predominantes en la foto son los colores claros (figura 10). Como podemos observar, tenemos una foto de un oso polar paseando por la nieve, por lo tanto, el histograma obtenido está formado por valores muy altos, muy próximos al color blanco.

Código ejemploHistogramBear.m

```
%Histogram Bear
image= ('Bear.jpg');
Im1 = imread(image);
Im=RGBtoGray(Im1);
title('Histograma Bear');
subplot(2,2,1),imshow(Im1),title('Imagen original
RGB'), xlabel('Largo'), ylabel('Alto');
subplot(2,2,2),imshow(Im),title('Imagen original Escala de
grises'), xlabel('Largo'), ylabel('Alto');
subplot(2,2,3),imhist(Im),title('Histograma'), xlabel('Rango de
grises'), ylabel('Número de píxeles');
```

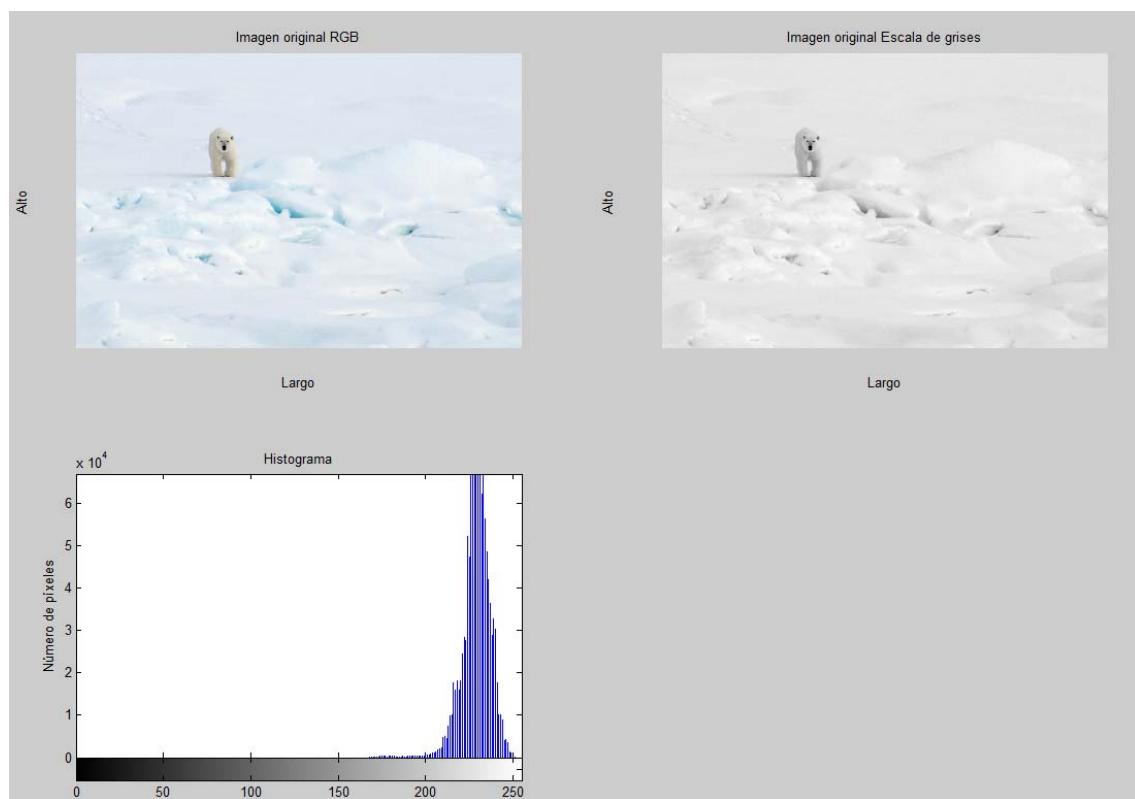


Figura 10. Histograma formado por colores claros.

### 3.1.3 Histograma estrecho.

El siguiente ejemplo muestra la definición teórica del contraste. Esta famosa propiedad mide la diferencia relativa en la intensidad entre un punto de la imagen y sus alrededores. Como podemos apreciar el contraste es muy bajo, ya que las diferencias entre colores muy claros y muy oscuros son pequeñas. Los valores dominantes son los intermedios “Histograma estrecho”.

Código ejemploHistogramFog.m

```
%Histogram Fog
image=('Fog.jpg');
Im1 = imread(image);
Im=RGBtoGray(Im1);
title('Histograma Fog');
subplot(2,2,1),imshow(Im1),title('Imagen original
RGB'), xlabel('Largo'), ylabel('Alto');
subplot(2,2,2),imshow(Im),title('Imagen original Escala de
grises'), xlabel('Largo'), ylabel('Alto');
subplot(2,2,3),imhist(Im),title('Histograma'), xlabel('Rango de
grises'), ylabel('Número de píxeles');
```

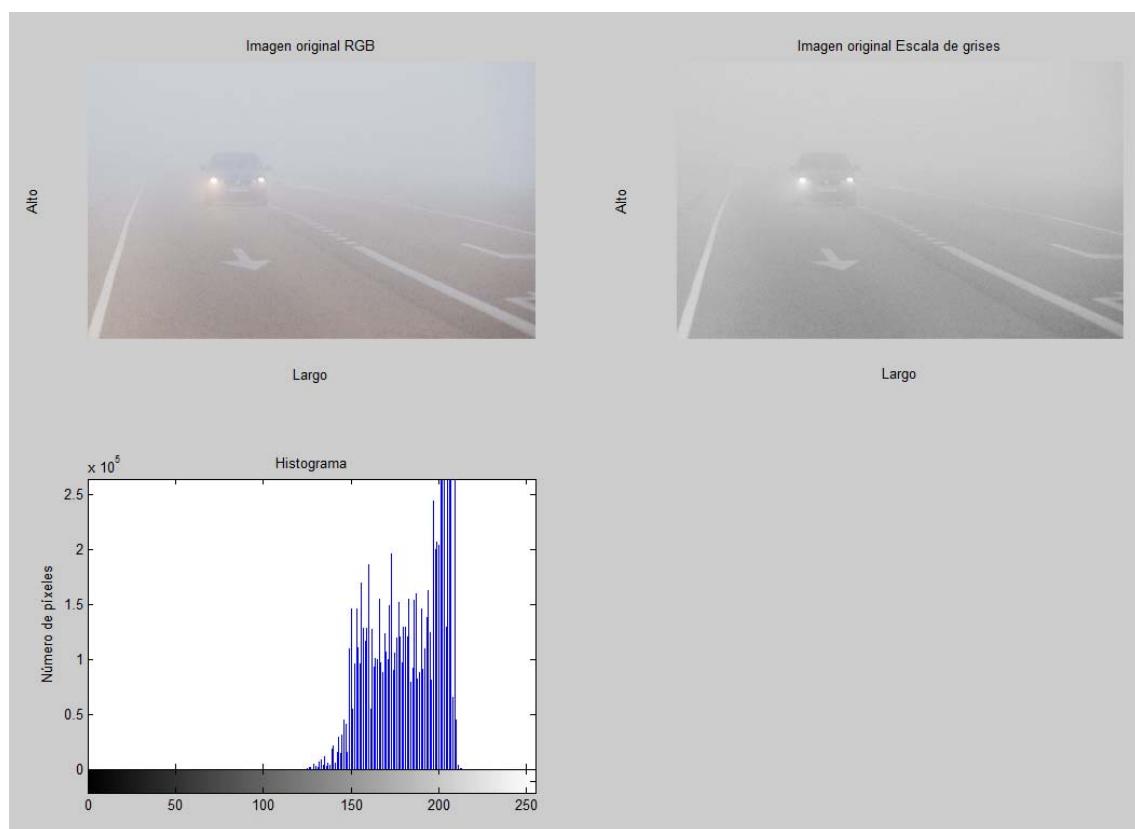


Figura 11. Histograma estrecho.

### 3.1.4 Histograma con gran contraste.

En conclusión, un histograma con valores muy bajos denota una imagen muy oscura, mientras que un histograma con valores muy altos denota una imagen muy clara. Para garantizar un buen procesado en el proyecto debemos obtener una imagen muy rica en el espectro de colores, es decir, ni muy clara, ni muy oscura, ni con un contraste muy bajo (ver histograma estrecho). La imagen idónea para el procesado es una imagen con gran contraste. La imagen utilizada en el ejemplo de la figura 12 es muy cómoda para conseguir un buen procesado, ya que, posee un contraste muy alto.

El tetrabrik tiene unos colores muy claros, mientras que el fondo es extremadamente oscuro (por no decir negro). Si tenemos un escenario real como éste, la identificación de figuras u objetos es muy sencilla mediante la técnica de histogramas. Lo único que tenemos que hacer es separar el objeto del fondo, en este caso, separar las componentes oscuras de las componentes claras y posteriormente comparar las imágenes.

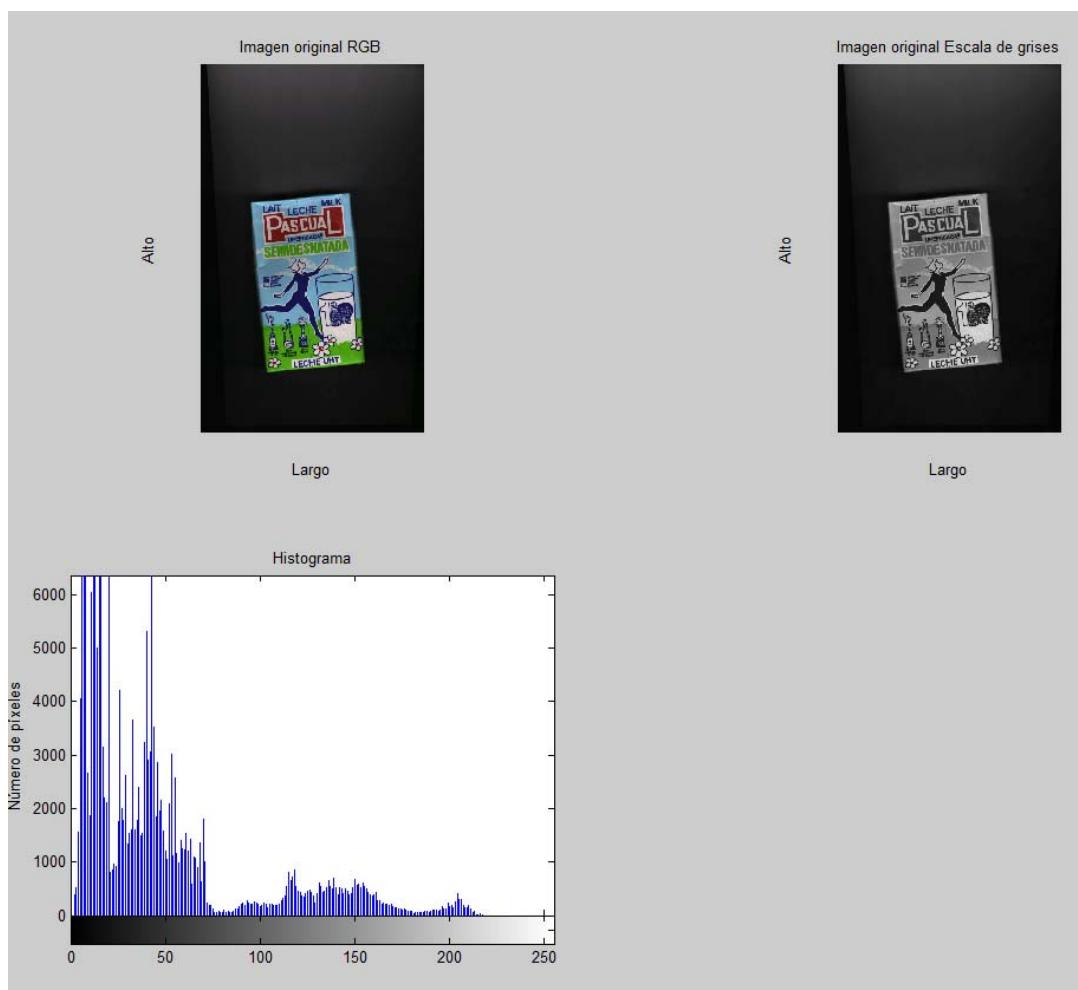


Figura 12. Histograma con gran contraste.

### Código ejemploHistogramContrast.m

```
%Histogram contrast
image = ('Tcol8.bmp');
Im1 = imread(image);
Im=RGBtoGray(Im1);
title('Histograma contrast');
subplot(2,2,1),imshow(Im1),title('Imagen original
RGB'), xlabel('Largo'), ylabel('Alto');
subplot(2,2,2),imshow(Im),title('Imagen original Escala de
grises'), xlabel('Largo'), ylabel('Alto');
subplot(2,2,3),imhist(Im),title('Histograma'), xlabel('Rango de
grises'), ylabel('Número de píxeles');
```

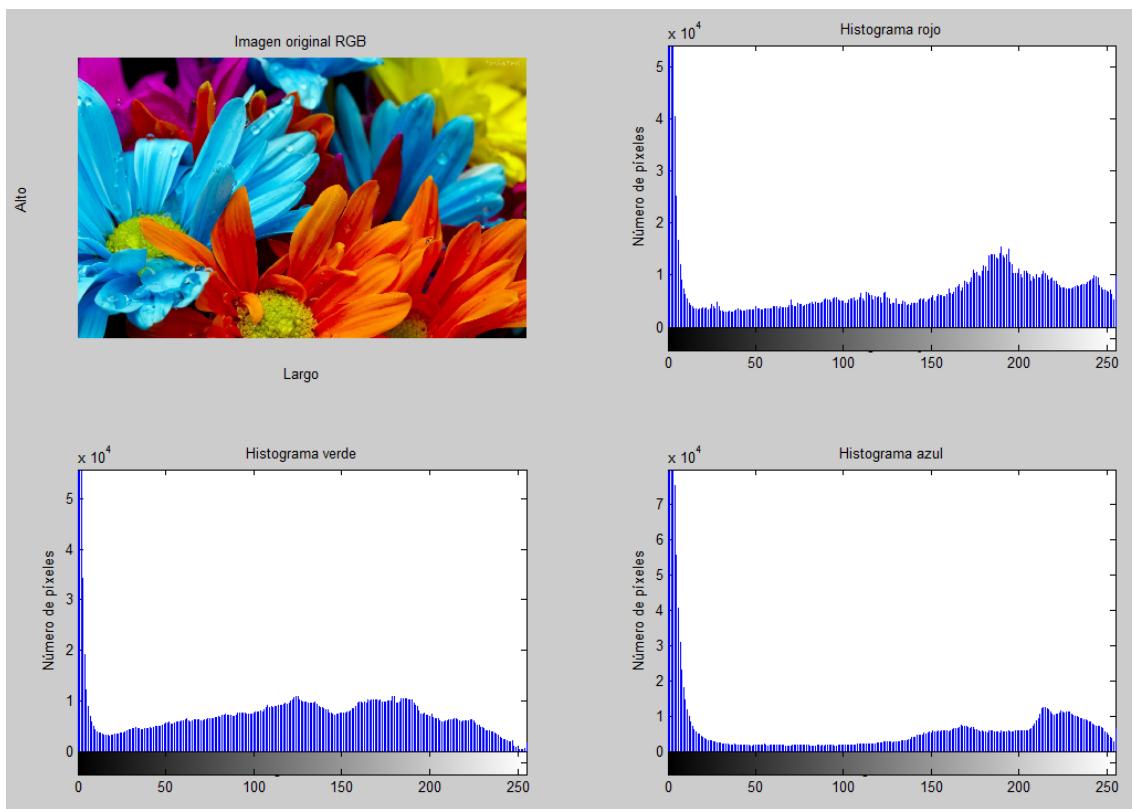
## 3.2 Histogramas a color.

En el apartado anterior hemos realizado todos los ejemplos con imágenes en la escala de grises. En este apartado realizaremos un análisis más detallado considerando que cada imagen está formada por la composición de tres canales de colores RGB. El procedimiento es muy parecido al del apartado anterior, simplemente debemos descomponer la imagen en cada uno de los canales de colores y realizar su correspondiente histograma.

### Código ejemploHistogramColorFlowers.m

```
%Histogram color Flowers
image = ('Flowers.jpg');
Im = imread(image);
Red = Im(:,:,1);
Green = Im(:,:,2);
Blue = Im(:,:,3);
title('Histograma Flowers');
subplot(2,2,1),imshow(Im),title('Imagen original
RGB'), xlabel('Largo'), ylabel('Alto');
subplot(2,2,2),imhist(Red),title('Histograma rojo'), xlabel('Rango de
rojos'), ylabel('Número de píxeles');
subplot(2,2,3),imhist(Green),title('Histograma verde'), xlabel('Rango
de verdes'), ylabel('Número de píxeles');
subplot(2,2,4),imhist(Blue),title('Histograma azul'), xlabel('Rango de
azules'), ylabel('Número de píxeles');
```

La eficiencia del uso de histogramas a color es muy cuestionable, depende claramente de lo que queremos identificar. La principal ventaja es que disponemos del comportamiento de los colores, siendo más eficaz la identificación de colores más concretos. Las desventajas son muchas, las imágenes son extremadamente dependientes de la iluminación escénica dificultándonos seriamente el estudio de los tonos de la imagen. Por otro lado, la identificación de los colores no es algo trivial pues dependen de las tres componentes RGB, convirtiéndose en una tarea complicada.



**Figura 13. Histograma de imagen a color.**

### 3.3 Algoritmo para la creación de un histograma HSL.

Una vez finalizado todos los ejemplos teóricos, nos disponemos a realizar un algoritmo para crear un histograma de las imágenes previamente convertidas a HSL. Es decir, vamos a continuar con el algoritmo que tenemos desarrollado para el proyecto. Vamos adaptar la técnica utilizada en los ejemplos de histogramas en la escala de grises al modelo HSL que hemos adoptado.

Como hemos dicho a lo largo del proyecto, un histograma es una representación gráfica de una variable en forma de barras, donde la superficie de cada barra corresponde a la frecuencia de los valores representados. Por lo tanto, el funcionamiento es similar al de un contador, vamos contando y almacenando el número de píxeles de cada tipo del modelo HSL.

El modelo HSL utiliza tres componentes *Hue*, *Saturation* y *Lightness*, como es obvio, representaremos un histograma para cada uno de los componentes.

El funcionamiento es el mismo para cada tipo de histograma (figura 14):

1. Recorremos la matriz de valores y extraemos el valor de un píxel.
2. Seleccionamos la posición del contador cuyo valor es equivalente al píxel.
3. Incrementamos el valor almacenado en dicha posición.
4. Repetimos el bucle hasta recorrer completamente la matriz de valores correspondiente.

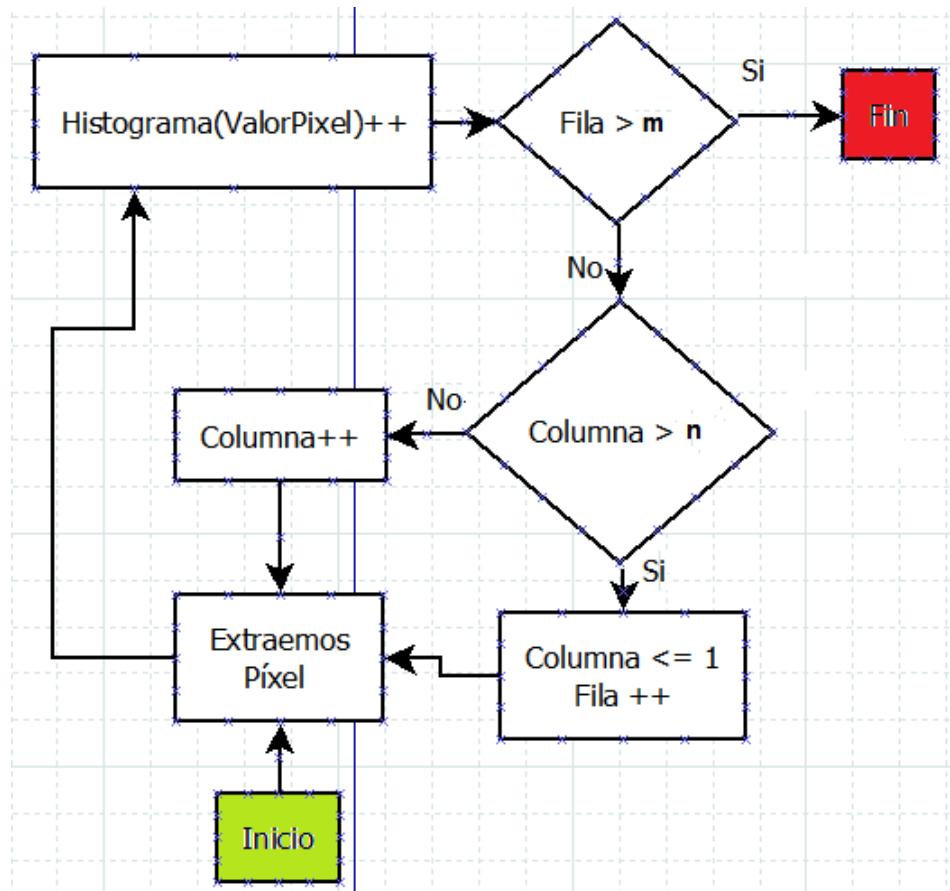


Figura 14. Diagrama de flujo del algoritmo de generación de histograma.

### Histograma Hue.

Para realizar el histograma Hue tenemos que almacenar los valores de los grados de cada color, por lo tanto, implementaremos un contador de 0 a 360.

### Histograma Saturation y Lightness.

Para realizar el histograma Saturation o Lightness tenemos que almacenar los porcentajes de tanto por ciento, por lo tanto, implantaremos un contador de 0 a 100.

Código ejemploHistogramHSL.m

```
%example HSL histogram
image=('Tcol8.bmp');
Im1 = imread(image);
[Hue,Saturation,Lightness]=RGB2HSL(Im1);
[histH,histL,hists]=histogram(Hue,Saturation,Lightness);
title('Histograma contrast');
subplot(2,2,1),imshow(Im1),title('Imagen original
RGB'), xlabel('Largo'), ylabel('Alto');
subplot(2,2,2),plot(histH),title('Hue
Histogram'), xlabel('Hue'), ylabel('Número de píxeles');
subplot(2,2,3),plot(hists),title('Saturation
Histogram'), xlabel('Saturation'), ylabel('Número de píxeles');
subplot(2,2,4),plot(histL),title('Lightness
Histogram'), xlabel('Lightness'), ylabel('Número de píxeles');
```

### Código ejemploHistogramHSL.m

```
%example HSL histogram
image=('Tcol8.bmp');
Im1 = imread(image);
[Hue,Saturation,Lightness]=RGB2HSL(Im1);
[histH,histL,hists]=histogram(Hue,Saturation,Lightness);
title('Histograma contrast');
subplot(2,2,1),imshow(Im1),title('Imagen original
RGB'), xlabel('Largo'), ylabel('Alto');
subplot(2,2,2),plot(histH),title('Hue
Histogram'), xlabel('Hue'), ylabel('Número de píxeles');
subplot(2,2,3),plot(hists),title('Saturation
Histogram'), xlabel('Saturation'), ylabel('Número de píxeles');
subplot(2,2,4),plot(histL),title('Lightness
Histogram'), xlabel('Lightness'), ylabel('Número de píxeles');
```

### Código Histogram.m

```
function [histH,histL,hists]=Histogram(H,S,L)
histH=Histogram360degrees(H);
histS=Histogram100Percent(S);
histL=Histogram100Percent(L);
end
```

### Código Histogram100Percent.m

```
function [histX]=Histogram100Percent(X)
[f,c]=size(X);
histX=zeros(101,1);
for i=1:f
for j=1:c
k = int16(X(i,j));
histX(k+1) = histX(k+1)+1;
end
end
```

### Código Histogram360degrees.m

```
function [histX]=Histogram360degrees (X)
[f,c]=size(X);
histX=zeros(361,1);
for i=1:f
for j=1:c
k = int16(X(i,j));
histX(k+1) = histX(k+1)+1;
end
end
```

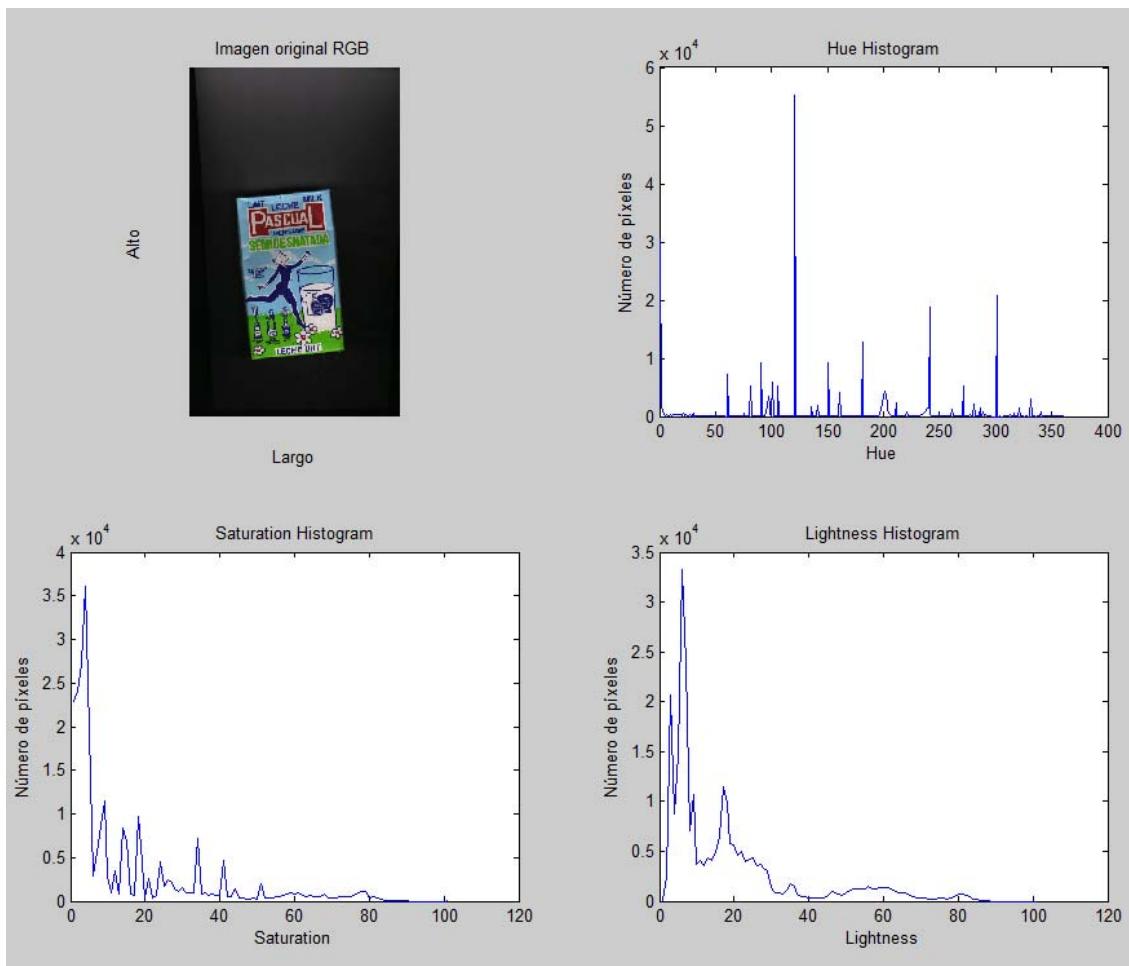


Figura 15. Histogramas resultantes al ejecutar el algoritmo de generación de histograma.

## 16. 4 Cálculo de distancias entre histogramas.

Finalmente después del trabajo realizado, comparamos los histogramas HSL obtenidos en los apartados anteriores. Para eso hacemos uso de las métricas estudiadas en el marco teórico.

La comparación entre el histograma de referencia y el muestreado nos permite la identificación de imágenes mediante la similitud de histogramas. El funcionamiento del

algoritmo seleccionado (*Chi-Cuadrado*) es un sumatorio de los errores obtenidos de comparar cada uno de los componentes del histograma de la imagen muestreada respecto al histograma de la imagen de referencia.

$$d(H_1, H_2) = \sum_i \frac{(H_2(i) - H_1(i))^2}{H_1(i) + H_2(i)}$$

El funcionamiento del algoritmo *Chi-Cuadrado* es el siguiente:

1. Recorremos el histograma de referencia y el histograma de muestra y extraemos el valor de  $H_1(i)$  y  $H_2(i)$  respectivamente.
2. Realizamos la operación:

$$\text{Error } i = \frac{(H_2(i) - H_1(i))^2}{H_1(i) + H_2(i)}$$

3. Incrementamos el valor del sumatorio:

$$d(H_1, H_2) = d(H_1, H_2) + \text{Error } i$$

4. Repetimos el bucle hasta recorrer completamente los histogramas.

A continuación se muestran, a modo de ejemplo, los resultados obtenidos al comparar los histogramas de dos imágenes muy similares (figura 16) y de dos imágenes muy diferentes (figura 17). En ambos casos se han utilizado la métrica *Chi-Cuadrado* y la *Bhattacharyya*.

- A) “tcol1.bmp” versus “tcol2.bmp”.



Figura 16. Comparación de imágenes similares.

-----tcol1.bmp vs tcol2.bmp-----

Bhattacharya

1. Distance Hue : 0.974986
2. Distance Saturation : 0.960334
3. Distance Ligthness : 0.982773
4. Media : 0.972698

Chi\_Square

- 1. Distance Hue : 0.008377**
2. Distance Saturation : 0.005863
3. Distance Ligthness : 0.003754
4. Media : 0.005998

-----  
B) “tcol1.bmp” versus “bear.jpg”.



**Figura 17. Comparación de imágenes muy diferentes.**

-----tcol1.bmp vs Bear.jpg-----

Bhattacharya

1. Distance Hue : 0.126139
2. Distance Saturation : 0.183354
3. Distance Ligthness : 0.362736

```

4. Media : 0.224076
Chi_Square
1. Distance Hue : 0.162384
2. Distance Saturation : 0.091217
3. Distance Ligthness : 0.111622
4. Media : 0.121741
-----
```

### Código CompareHistograms.m

```

% read two images
imagen1=('tcol1.bmp');
imagen2=('tcol2.bmp');
Im1 = imread(imagen1);
Im2 = imread(imagen2);

% convert RGB to HSL
[H1,S1,L1]=rgb2hsl(Im1);
[H2,S2,L2]=rgb2hsl(Im2);

[m,n]=size(H1);
numpixels1=m*n;
[m,n]=size(H2);
numpixels2=m*n;

% Calculate histograms

[histH1,histL1,histS1]=histograma(H1,L1,S1);
[histH2,histL2,histS2]=histograma(H2,L2,S2);

%normalized histograms

histH1Normalizado=histH1./numpixels1;
histH2Normalizado=histH2./numpixels2;
histS1Normalizado=histS1./numpixels1;
histS2Normalizado=histS2./numpixels2;
histL1Normalizado=histL1./numpixels1;
histL2Normalizado=histL2./numpixels2;

%Bhattacharya distance
bH=Bhattacharya(histH1Normalizado,histH2Normalizado);
bS=Bhattacharya(histS1Normalizado,histS2Normalizado);
bL=Bhattacharya(histL1Normalizado,histL2Normalizado);

%Chi_Square distance
cH=Chi_Square(histH1Normalizado,histH2Normalizado);
cS=Chi_Square(histS1Normalizado,histS2Normalizado);
cL=Chi_Square(histL1Normalizado,histL2Normalizado);

Mediab=(bH+bS+bL)/3;
```

```
Mediac= (cH+cS+cL) / 3;
```

```
fprintf('-----');
fprintf(imagen1);
fprintf(' vs ');
fprintf(imagen2);
fprintf('-----\n');

fprintf('Bhattacharya\n');
fprintf('1. Distance Hue : %f\n',bH);
fprintf('2. Distance Saturation : %f\n',bS);
fprintf('3. Distance Ligthness : %f\n',bL);
fprintf('4. Media : %f\n',Mediac);

fprintf('Chi_Square\n');
fprintf('1. Distance Hue : %f\n',cH);
fprintf('2. Distance Saturation : %f\n',cS);
fprintf('3. Distance Ligthness : %f\n',cL);
fprintf('4. Media : %f\n',Mediac);

fprintf('-----\n');
```

### Código Chi\_Square.m

```
function [cdist]=Chi_Square(histogram1,histogram2)
bins= size(histogram1,2);
cdist=0;
for i=1:bins
    cdist=cdist+((histogram1(i)-histogram2(i))^2)/(histogram1(i)+histogram2(i));
end
end
```

Como prueba experimental añado el código para el cálculo de la distancia entre histogramas utilizando la métrica **Bhattacharyya**.

### Código Bhattacharyya.m

```
function [bdist]=Bhattacharya(histogram1,histogram2)
bdist = sum(sum(sqrt(histogram1).*sqrt(histogram2)));
end
```



## ANEXO 5: Implementación hardware del sistema

### Contenido

1.	Introducción.....	96
2.	Esquema general. ....	96
2.1	Esquema RTL del sistema completo.....	97
2.2	Recursos utilizados. ....	98
2.3	Análisis de frecuencia. ....	99
3.	Algoritmo para la conversión imágenes RGB a HSL.....	99
3.1	Esquema general RGB a HSL. ....	99
3.1.1	Esquema RTL. ....	100
3.1.2	Simulación Post-Route.....	101
3.1.3	Recursos utilizados:....	102
3.1.4	Análisis de frecuencia. ....	103
3.2	Módulo Máximo y mínimo. ....	103
3.2.1	Esquema RTL. ....	104
3.2.2	Simulación Post-Route.....	104
3.2.3	Recursos utilizados. ....	105
3.2.4	Análisis de frecuencia. ....	106
3.3	Módulo C. ....	106
3.3.1	Esquema RTL. ....	106
3.3.2	Simulación Post-Route.....	106
3.3.3	Recursos utilizados. ....	107
3.3.4	Análisis de frecuencia. ....	108
3.4	Módulo Hue. ....	109
3.4.1	Esquema RTL. ....	110
3.4.2	Simulación Post-Route.....	111
3.4.3	Recursos utilizados. ....	112

3.4.4	Análisis de frecuencia .....	112
3.5	Módulo Saturation .....	113
3.5.1	Esquema RTL .....	114
3.5.2	Simulación Post-Route.....	115
3.5.3	Recursos utilizados.....	116
3.5.4	Análisis de frecuencia .....	116
3.6	Módulo Lightness .....	117
3.6.1	Esquema RTL .....	118
3.6.2	Simulación Post-Route.....	119
3.6.3	Recursos utilizados.....	120
3.6.4	Análisis de frecuencia .....	120
4.	Algoritmo para la creación de un histograma HSL .....	121
4.1	Esquema RTL .....	122
4.2	Simulación Post-Route.....	123
4.3	Recursos utilizados .....	124
4.4	Análisis de frecuencia .....	124
5.	Cálculo de distancias entre histogramas, Chi-Cuadrado .....	124
5.1	Esquema general .....	124
5.1.1	Esquema RTL .....	125
5.2	Submódulo Chi-cuadrado.....	126
5.2.1	Esquema RTL.....	126
5.2.2	Simulación Post-Route.....	127
5.2.3	Recursos utilizados .....	128
5.2.4	Análisis de frecuencia .....	128

## 1. Introducción.

En este anexo, añadimos la documentación técnica del sistema realizado en la FPGA. Se incluyen los esquemas RTL de los bloques diseñados, los resultados de simulación e información sobre los recursos utilizados y las características de frecuencias.

El resultado del TFG queda concluido con el diseño en VHDL de la aplicación desarrollada para la FPGA Spartan 6 [11][12].

## 2. Esquema general.

El diseño final del proyecto queda de la siguiente forma:

- Módulo conversor RGB a HSL: Direcciona los píxeles de entrada que son recibidos del módulo VmodCAM y los convierte del modelo de color RGB al modelo de color HSL.
- Módulo histograma: Va creando el histograma a medida que va recibiendo cada píxel en el modelo de color HSL. El histograma queda completamente finalizado una vez recibido todos los píxeles que forman la imagen capturada.
- Comparador Chi-cuadrado: Calcula la distancia geométrica entre el histograma de referencia almacenado y el histograma muestreado de la imagen capturada.

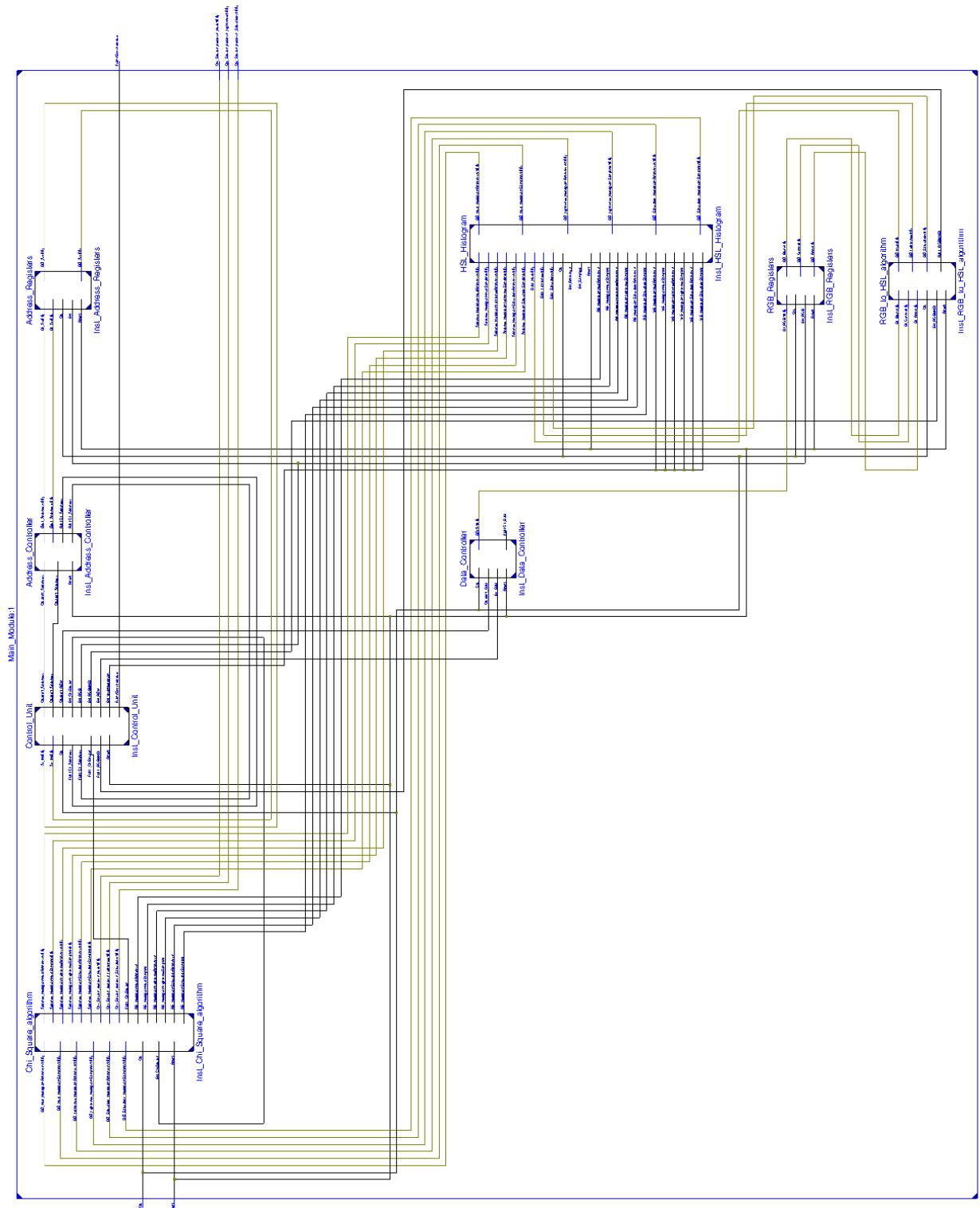
El esquema general presentado (ver apartado 2.1), no muestra la conexión establecida con el módulo VmodCAM. La entrada de los píxeles del módulo VmodCAM es simulada mediante un bloque que indica la posición del píxel en la matriz de la imagen (*Address\_Controller* y *Address\_Registers*) y un bloque que simula la entrada de la trama de bits que está formada por los píxeles RGB (*Data\_Controller*). La trama de 16 bits recibida del módulo VmodCAM, es descompuesta en cada una de las componentes que forman el modelo RGB mediante el bloque *RGB\_Registers* (ver figura 1).



Figura 3: Trama de bits recibida del módulo VmodCAM.

El funcionamiento global del proyecto es controlado por una unidad que controla que gestiona la simulación del módulo VmodCAM, el conversor RGB a HSL, el módulo histograma y el comparador *Chi-cuadrado*.

## 2.1 Esquema RTL del sistema completo.



## 2.2 Recursos utilizados.

El diseño del proyecto implementado tiene un consumo de los siguientes recursos de la FPGA:

Device Utilization Summary					[-]
Slice Logic Utilization	Used	Available	Utilization	Note(s)	
Number of Slice Registers	9,825	54,576	18%		
Number of Slice LUTs	5,008	27,288	18%		
Number of occupied Slices	3,405	6,822	49%		
Number of MUXCYs used	556	13,644	4%		
Number of LUT Flip Flop pairs used	11,493				
Number of bonded IOBs	60	218	27%		
Number of RAMB16BWERS	5	116	4%		
Number of BUFG/BUFGMUXs	1	16	6%		
Number of DSP48A1s	44	58	75%		
Average Fanout of Non-Clock Nets	3.41				

En los siguientes párrafos se analiza porqué la FPGA hace uso de estos recursos para implementar la solución del proyecto, o dicho de otra manera, qué módulos son los que posibilitan un procesado de señal rápido y eficaz en una FPGA.

1. Slice Registers: Biestables, latches y AND/OR lógicas.
2. Slice LUTs: RAM distribuida, compite por recursos con la lógica. Su uso está restringido a aplicaciones que requieran poco almacenamiento.
3. LUT Flip Flop y Slices: Es la lógica distribuida utilizada de la FPGA.
4. RAMB16BWERS: Es un bloque dedicado de memoria, independiente de los bloques lógicos.
5. IOBs: Es el número de pines que está utilizando el dispositivo, entradas y salidas.
6. DSP48A1: Es un DSP diseñado teniendo en cuenta las tareas más habituales del procesado digital: sumas, multiplicaciones y acumulaciones (almacenamiento en memoria). Este bloque interno de la FPGA mejora la utilidad, la versatilidad y la velocidad de los bloques aritméticos diseñados. Una de las características más importantes es la capacidad de conectar en cascada módulos DSP48A1, es decir, conectar la salida de un DSP a la entrada del siguiente DSP, permitiéndonos grandes rendimientos y bajos consumos.
7. BUFG/BUFGMUXs: Para el diseño de señales globales, se usan buffers para mantener el *skew* bajo y una gran difusión por el circuito. Este el caso del reloj general utilizado a lo largo del circuito.

8. **MUXCYs:** Es utilizado para implementar desplazamientos lógicos con un 1 bit de acarreo.

## 2.3 Análisis de frecuencia.

Timing Summary:

---

Speed Grade: -3

Minimum period: 21.527ns (Maximum Frequency: 46.454MHz)

Minimum input arrival time before clock: 7.170ns

Maximum output required time after clock: 4.895ns

Maximum combinational path delay: No path found

La frecuencia de reloj con la que podemos trabajar es de 46 MHz, una frecuencia baja teniendo en cuenta que cada trama de 16 bits del módulo *VmodCAM* es recibida cada 25MHz. Esta es la razón principal por la que usaremos técnicas para aumentar la velocidad de procesamiento, como *pipeline*, en algunos de los módulos implementados en la FPGA.

## 17. 3. Algoritmo para la conversión imágenes RGB a HSL.

### 3.1 Esquema general RGB a HSL.

El módulo *RGB\_to\_HSL*, convierte el píxel RGB recibido en la trama 16 bits, al modelo de color HSL. Para realizar dicha operación realizamos los siguientes pasos (ver figura 2).

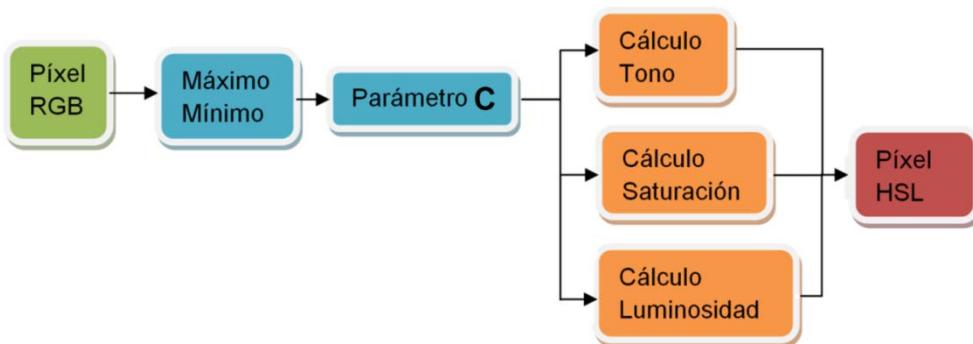


Figura 4: Algoritmo de conversión de RGB a HSL

1. Extraemos el valor de la componente RGB del píxel cuyo valor es máximo y mínimo.

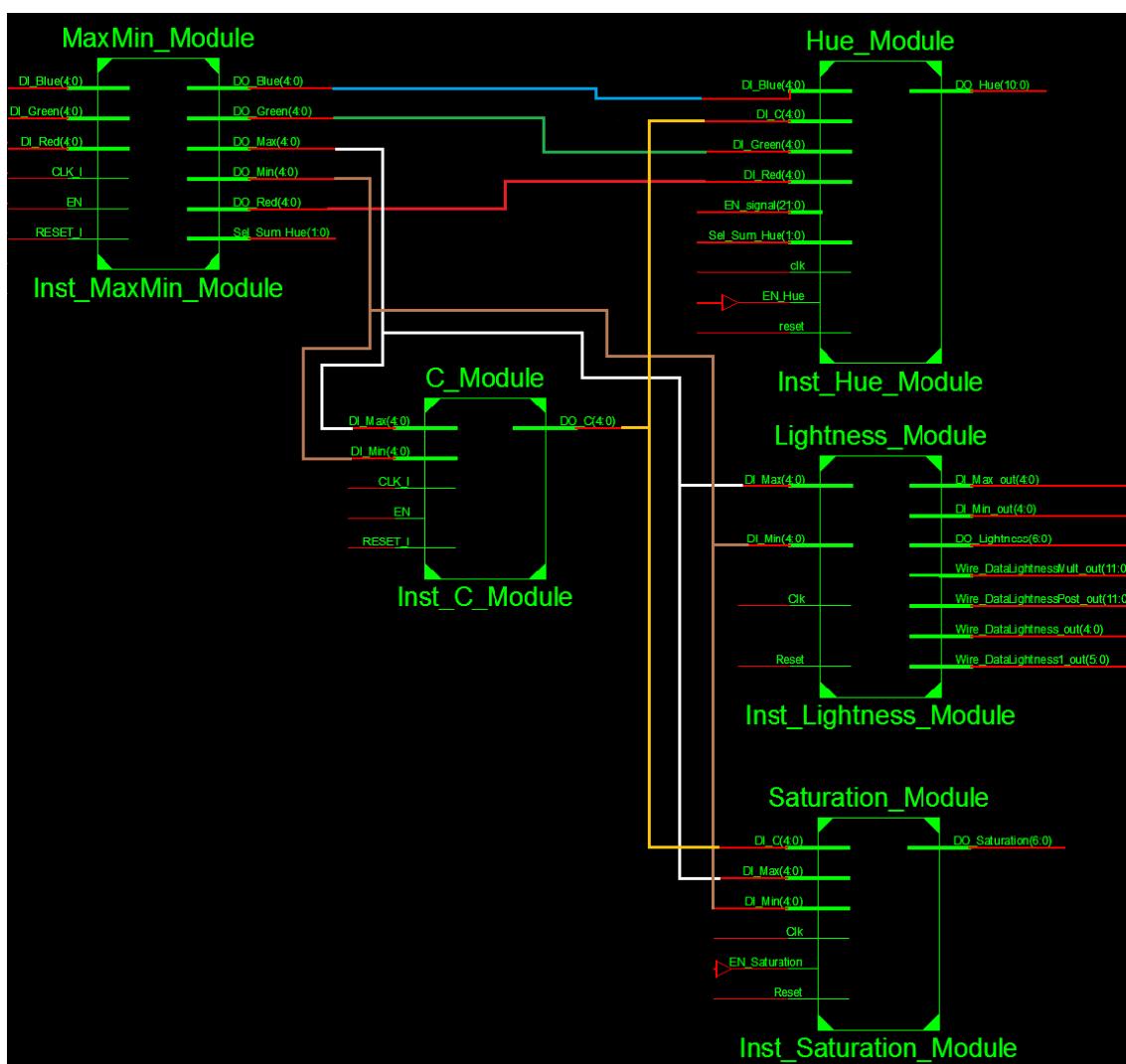
$$M = \max(R(i,j), G(i,j), B(i,j))$$
$$m = \min(R(i,j), G(i,j), B(i,j))$$

2. Calculamos el parámetro C.

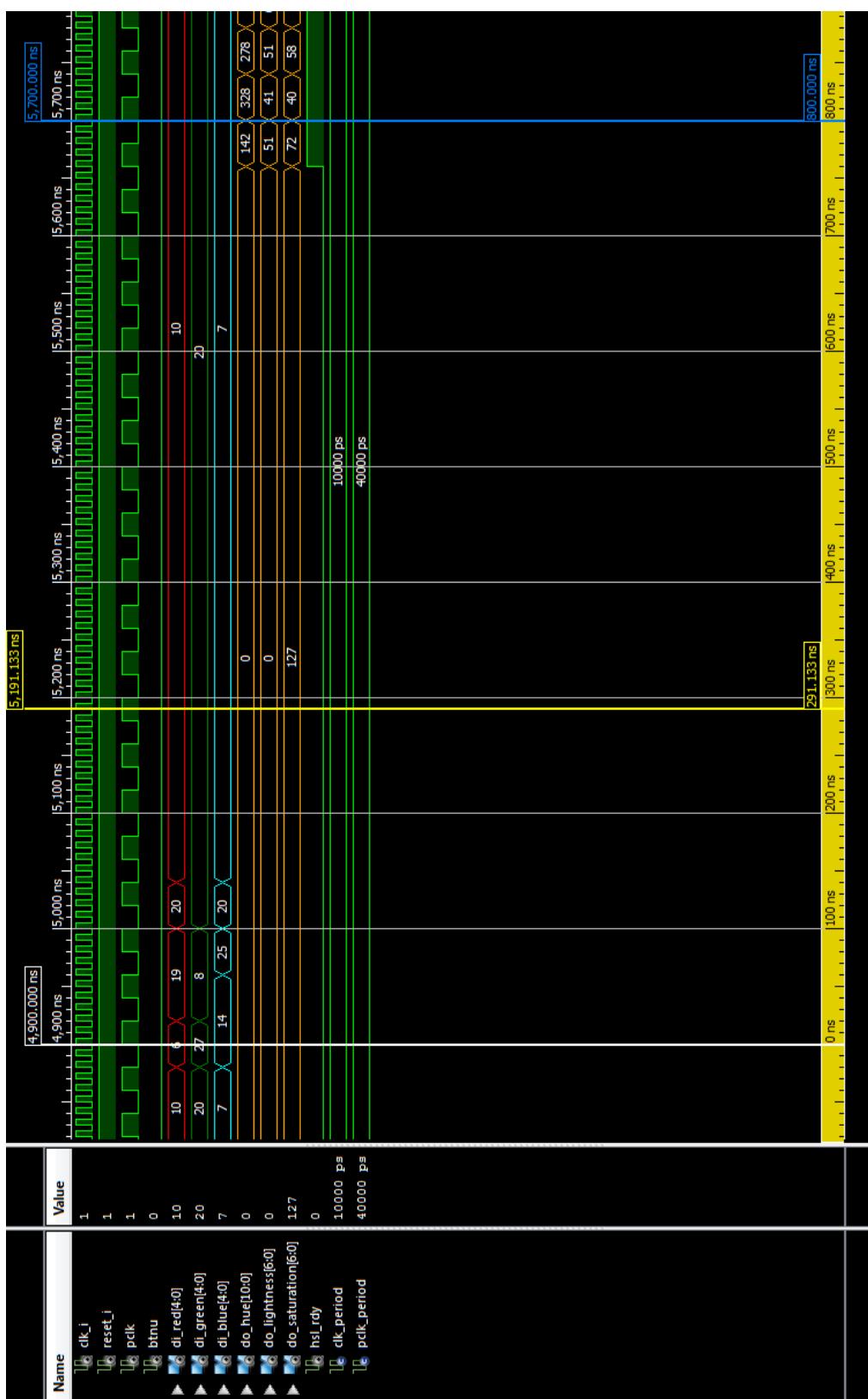
$$C = M - m$$

3. Aplicamos el algoritmo respectivo para el cálculo del tono, saturación o luminosidad.

### 3.1.1 Esquema RTL.



### 3.1.2 Simulación Post-Route.



### 3.1.3 Recursos utilizados:

Para el implementar el conversor RGB a HSL son necesarios los siguientes bloques de la FPGA.

1. 882 registros, de los cuales muchos de ellos son utilizados para la técnica *pipeline*.
2. 1 Buffer BUFG para evitar el *skew* de señales generales, como el reloj.
3. 92 pines para entradas y salidas.
4. 3 DSP48A1s para el cálculo de operaciones de procesado de señal.
5. 291+876 Slices y LUT Flip Flop, para elaborar la lógica distribuida.
6. 497 Slices LUTs, utilizados para el cálculo de divisiones.

### 3.1.4 Análisis de frecuencia.

Timing Summary:

Speed Grade: -3

Minimum period: 5.441ns (Maximum Frequency: 183.773MHz)

Minimum input arrival time before clock: 6.280ns

Maximum output required time after clock: 4.521ns

Maximum combinational path delay: No path found

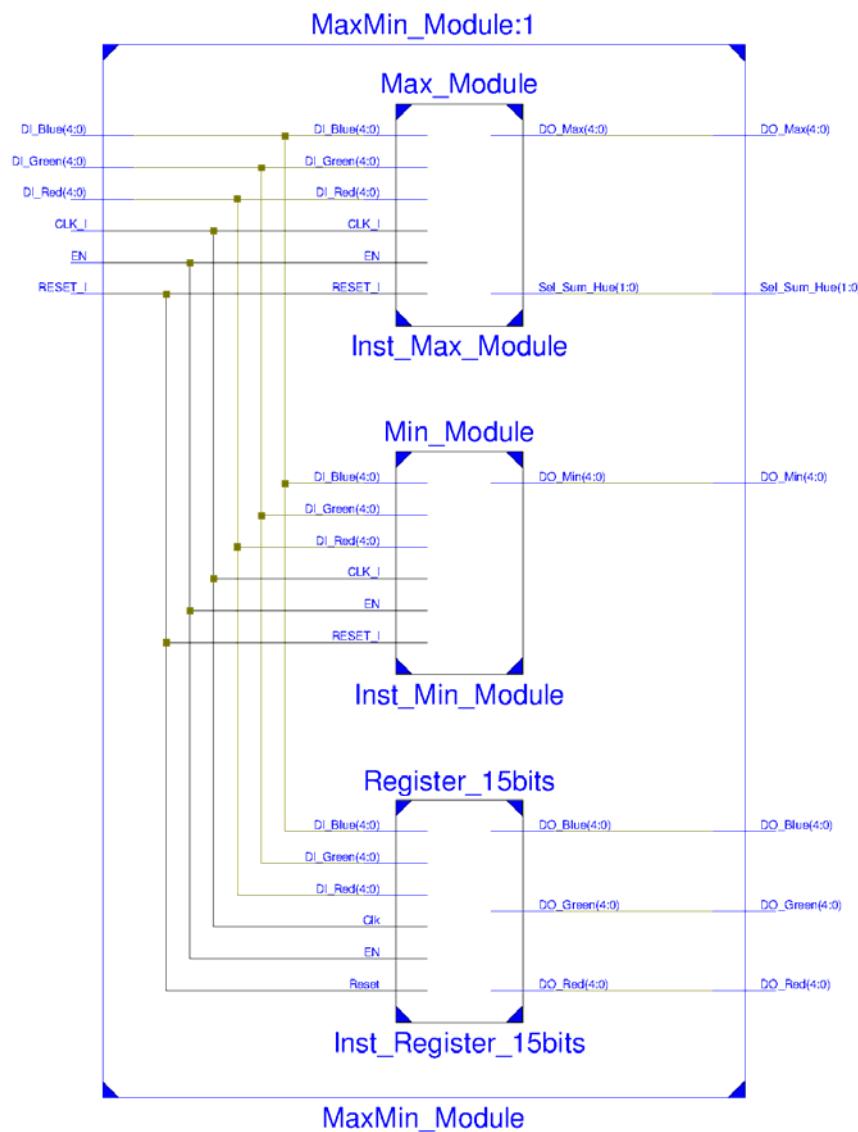
Como he dicho en el apartado 2.3, una de las técnicas utilizadas para la optimización temporal será el *pipeline*. Técnica indispensable en nuestro proyecto si no queremos perder ninguna información recibida por el módulo *VmodCAM*, ya que la velocidad de éste es de 25 MHz.

La ruta crítica del conversor RGB a HSL será la frecuencia máxima de trabajo alcanzada, 183Mhz (ver apartado 3.1.2). Mediante la segmentación equitativa, el cálculo total del algoritmo de conversión fija como frecuencia óptima la frecuencia máxima de trabajo, a costa de más registros entre los datos intermedios. La consecuencia será un aumento de la latencia (en ciclos/tiempo) en la salida del resultado, equivalente al número de registros implementados. La ventaja del sistema es que una vez que el *pipe* (tubo) está lleno, después de una latencia de 800ns, los resultados siguientes se obtienen uno tras otro, cada un ciclo de reloj y sin latencia extra, por estar encadenados dentro del mismo *pipe*.

## 3.2 Módulo Máximo y mínimo.

Para extraer el valor máximo y el valor mínimo de las componentes RGB de un píxel, utilizamos circuitos combinacionales llamados comparadores.

### 3.2.1 Esquema RTL.



### 3.2.2 Simulación Post-Route.



Como podemos apreciar en la simulación, el circuito combinacional extrae el valor máximo y mínimo de las componentes RGB del píxel de entrada.

$$RGB\{6,27,14\} \text{ máximo} = 27, \text{mínimo} = 6$$

### 3.2.3 Recursos utilizados.

Para el implementar el detector de máximos y mínimos son necesarios los siguientes bloques de la FPGA.

1. 2 registros, utilizados para hacer *pipeline* de la señal RGB.
2. 45 pines para entradas y salidas.
3. 12+27 Slices y LUT Flip Flop, para elaborar la lógica distribuida.
4. 27 Slices LUTs, utilizados para implementar el circuito combinacional comparador.

### 3.2.4 Análisis de frecuencia.

Timing Summary:

Speed Grade: -3

Minimum period: 1.371ns (Maximum Frequency: 729.262MHz)

Minimum input arrival time before clock: 6.280ns

Maximum output required time after clock: 3.634ns

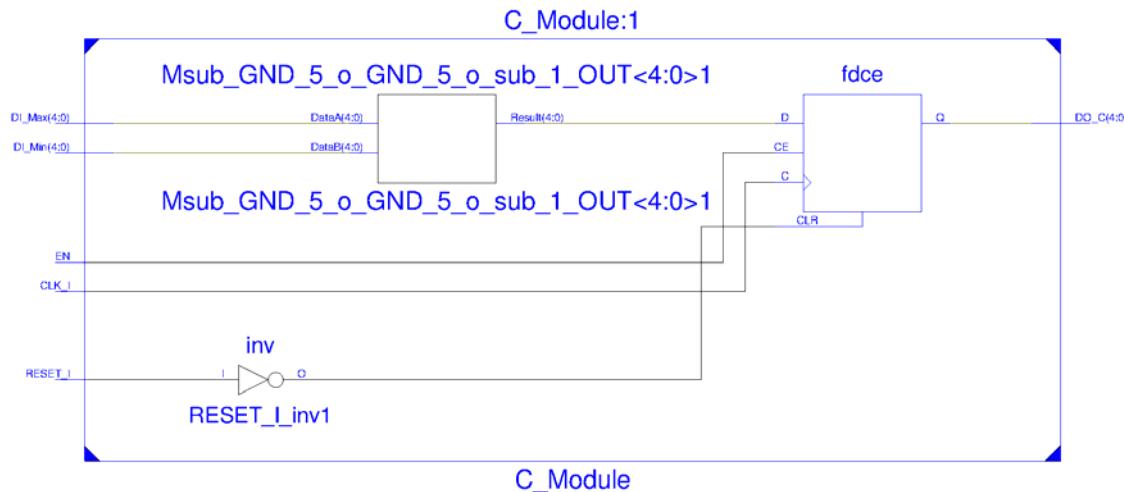
Maximum combinational path delay: No path found

Al usar la técnica *pipeline*, la frecuencia máxima de trabajo disminuye drásticamente. Un circuito combinacional es totalmente asíncrono.

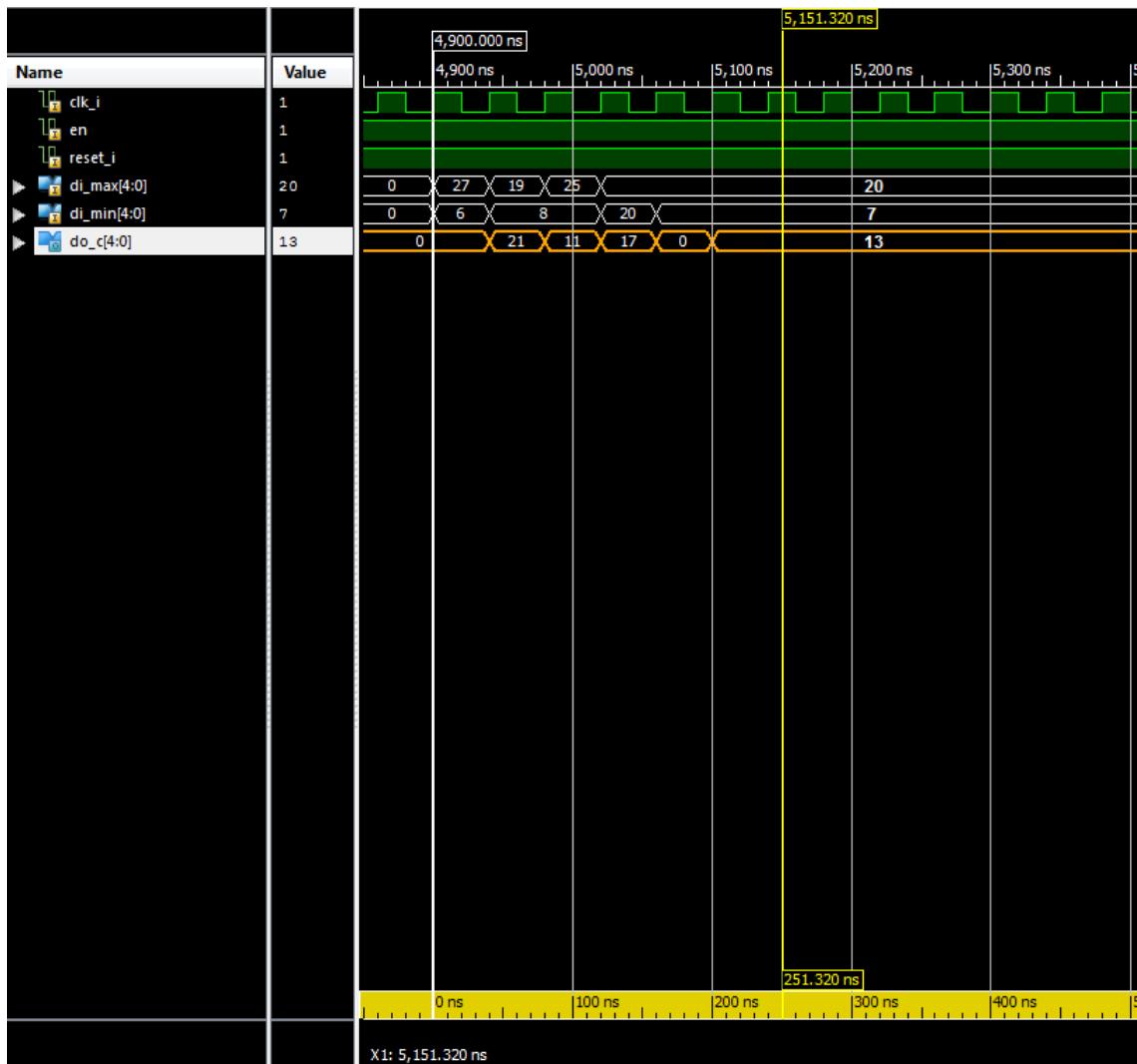
### 3.3 Módulo C.

Implementamos un sumador para el máximo y el mínimo, y realizamos un desplazamiento lógico hacia la derecha para obtener la división entre 2.

#### 3.3.1 Esquema RTL.



#### 3.3.2 Simulación Post-Route.



Como podemos apreciar en la simulación, el circuito implementado suma los valores máximo y mínimo y desplaza hacia la derecha.

- 1.
- 2.
3. Desplazamos

### 3.3.3 Recursos utilizados.

Para el implementar el cálculo del parámetro C son necesarios los siguientes bloques de la FPGA.

1. 18 pines para entradas y salidas.
2. 3 Slices, para elaborar la lógica distribuida.
3. 6 Slices LUTs, utilizados para implementar el circuito combinacional sumador y el desplazamiento lógico.

### 3.3.4 Análisis de frecuencia.

Timing Summary:

-----

Speed Grade: -3

Minimum period: No path found

Minimum input arrival time before clock: 3.359ns

Maximum output required time after clock: 3.597ns

Maximum combinational path delay: No path found

Al ser un circuito combinacional es totalmente asíncrono, por lo tanto, no afecta a la velocidad del diseño implementado.

## 3.4 Módulo Hue.

Las ecuaciones usadas son las siguientes:

$$H(i,j) = 0 \text{ si } M = m$$

$$H(i,j) = \text{mod} \left( 60 * \frac{G(i,j) - B(i,j)}{C} + 360, 360 \right) \text{ si } M = R(i,j)$$

$$H(i,j) = 60 * \frac{B(i,j) - R(i,j)}{C} + 120 \text{ si } M = G(i,j)$$

$$H(i,j) = 60 * \frac{R(i,j) - G(i,j)}{C} + 240 \text{ si } M = B(i,j)$$

Como vamos a implementarlo en una FPGA tenemos que hacer un diseño totalmente secuencial. Los módulos a diseñar son los siguientes. (Ver figura 3).

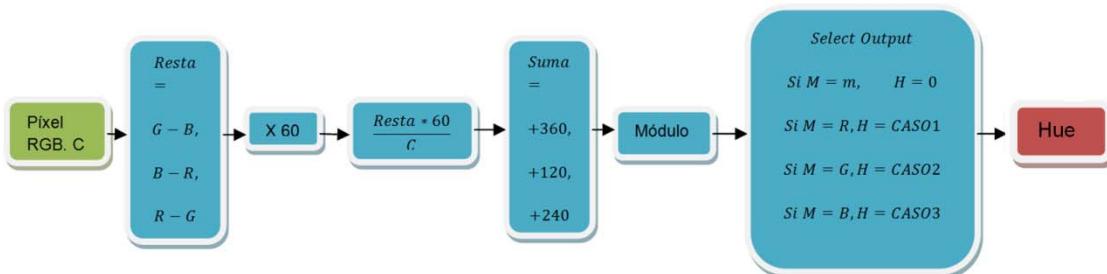
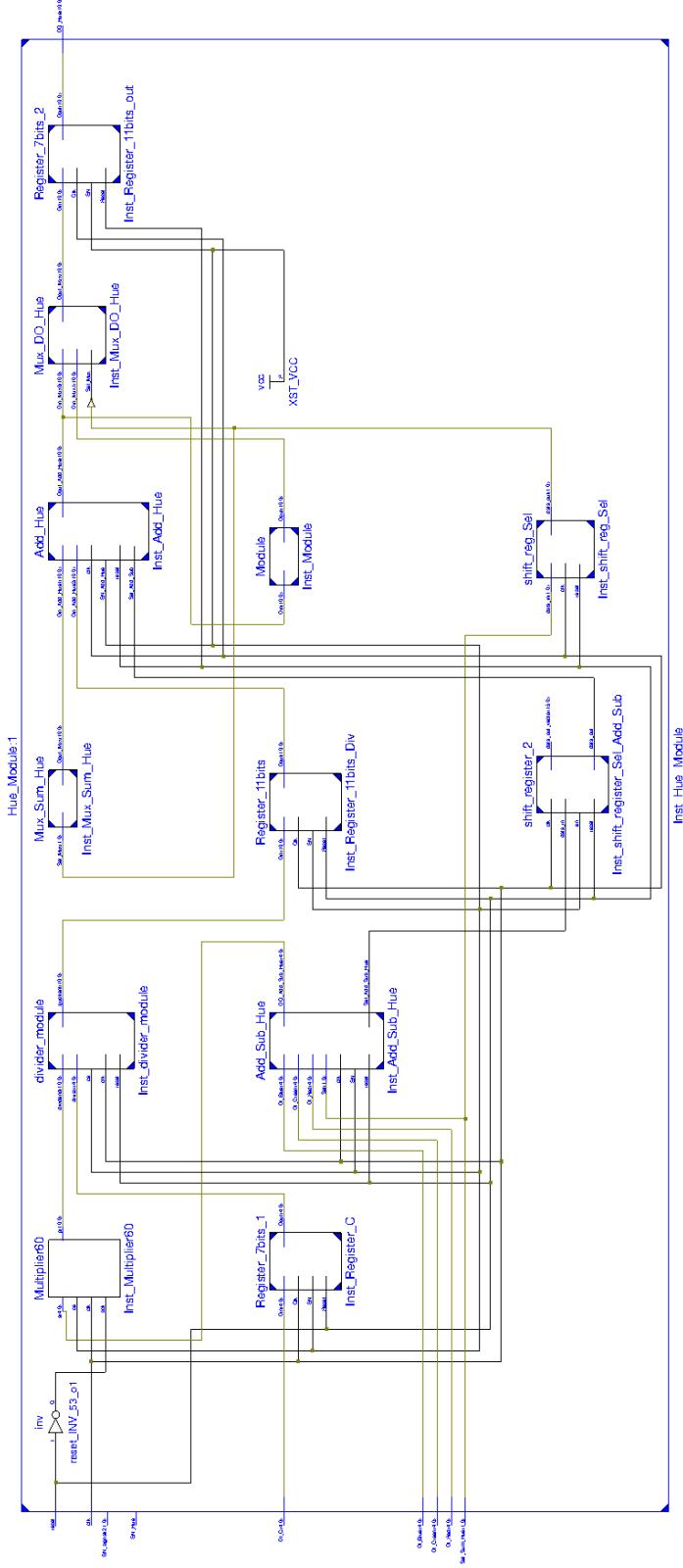


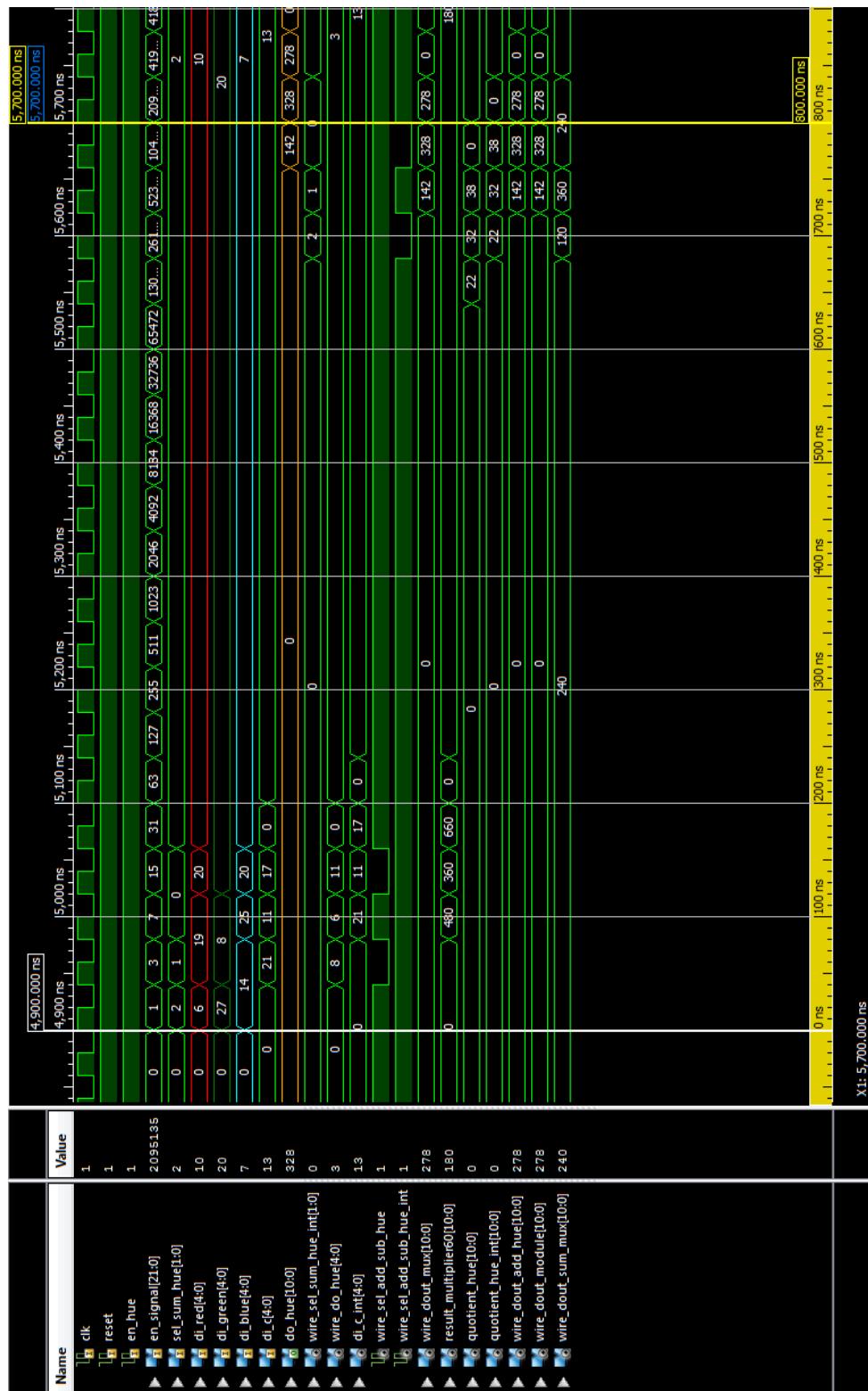
Figura 5: Algoritmo Hue.

No debemos olvidar que el orden debe ser estrictamente el siguiente: Primero multiplicamos y después dividimos. La razón es la siguiente, si dividimos primero el resultado quedaría normalizado a 1, valor no representable mediante un *unsigned\_std\_logic*.

### 3.4.1 Esquema RTL.



### 3.4.2 Simulación Post-Route



### 3.4.3 Recursos utilizados.

Para el implementar el algoritmo *Hue* son necesarios los siguientes bloques de la FPGA.

1. 882 registros, de los cuales muchos de ellos son utilizados para la técnica *pipeline*.
2. 92 pines para entradas y salidas.
3. 3 DSP48A1s para el cálculo de operaciones de procesado de señal.
4. 291+876 Slices y LUT Flip Flop, para elaborar la lógica distribuida.
5. 497 Slices LUTs, configurados como divisor.

### 3.4.4 Análisis de frecuencia.

Timing Summary:

---

Speed Grade: -3

Minimum period: 4.980ns (Maximum Frequency: 200.809MHz)

Minimum input arrival time before clock: 8.798ns

Maximum output required time after clock: 3.597ns

Maximum combinational path delay: No path found

La ruta crítica del algoritmo *Hue* quedará determinada por el CASO1, que es el peor de los casos (el que posee más cuentas). Al utilizar *pipeline* añadimos registros intermedios para que el cálculo total del algoritmo de *Hue* fije como frecuencia óptima la frecuencia máxima de trabajo, 200Mhz.

## 3.5 Módulo Saturation.

Las ecuaciones utilizadas son las siguientes:

$$\text{Saturation} = 0 \text{ si Lightness} = 0$$

$$\text{Saturation} = \frac{100 * C}{M + m} \text{ si Lightness} < 50$$

$$\text{Saturation} = \frac{100 * C}{2 - M - m} \text{ si Lightness} \geq 50$$

Como vamos a implementarlo en una FPGA tenemos que hacer un diseño totalmente secuencial. Los módulos a diseñar son los siguientes. (Ver figura 4).

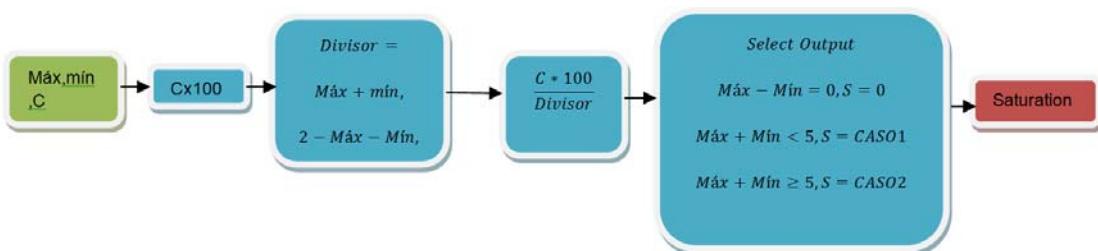
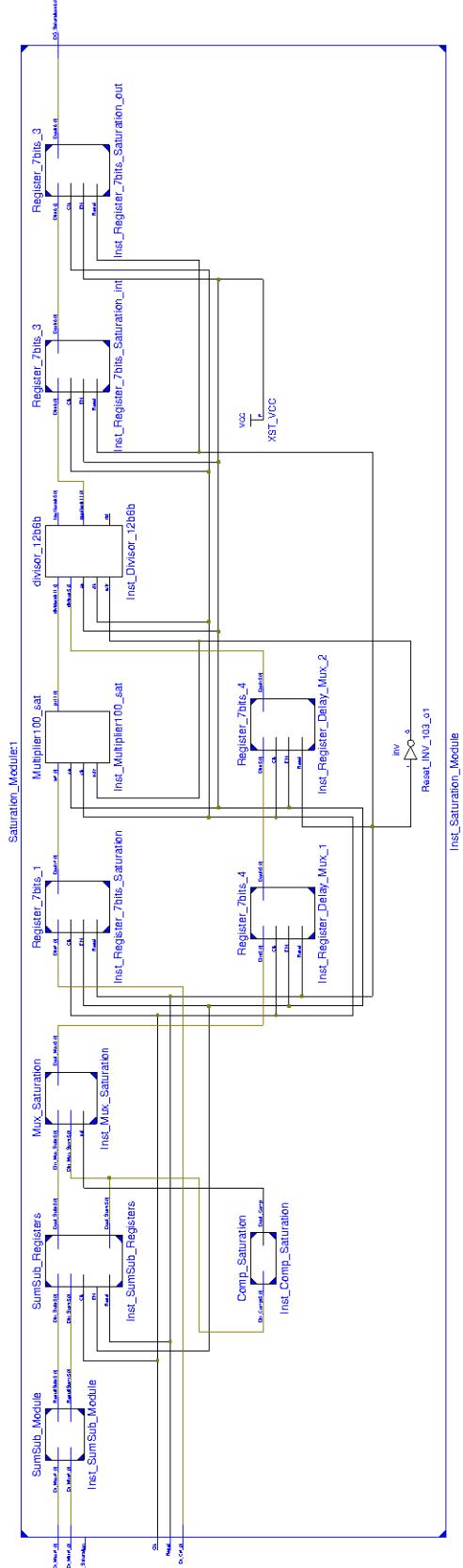


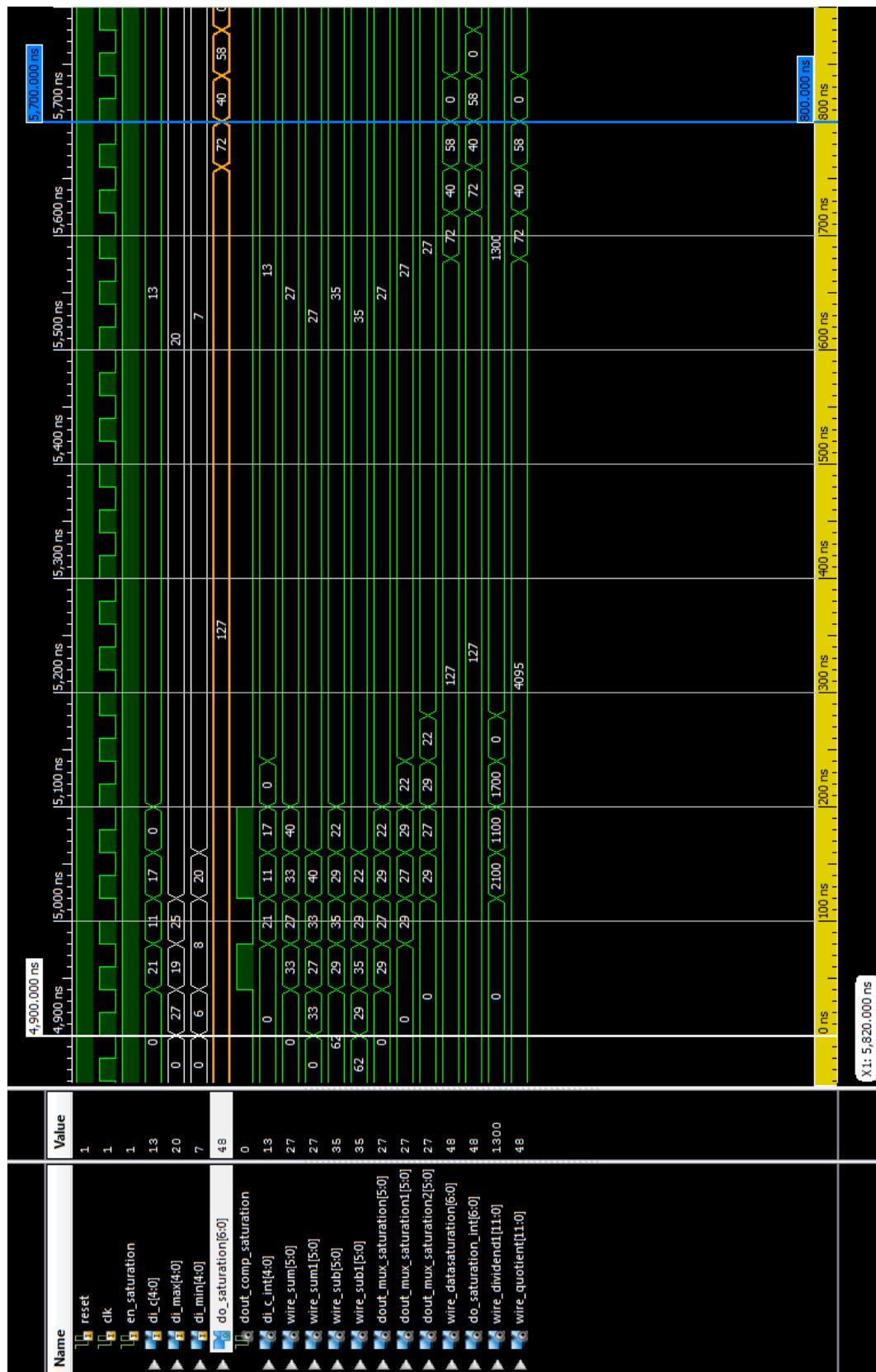
Figura 6: Algoritmo Saturation.

Para obtener un resultado en la escala adecuada, el valor de C debe ser multiplicado por 100. El motivo es que si dividimos el valor de C entre el denominador el resultado quedaría normalizado a 1, valor no representable en *unsigned\_std\_logic*.

### 3.5.1 Esquema RTL.



### 3.5.2 Simulación Post-Route.



### 3.5.3 Recursos utilizados.

Para el implementar el algoritmo *Saturation* son necesarios los siguientes bloques de la FPGA.

1. 260 registros, de los cuales muchos de ellos son utilizados para la técnica *pipeline*.
2. 24 pines para entradas y salidas.
3. 1 DSP48A1s para el cálculo de operaciones de procesado de señal.
4. 60+199 Slices y LUT Flip Flop, para elaborar la lógica distribuida.
5. 172 Slices LUTs, configurados como divisor.

### 3.5.4 Análisis de frecuencia.

Timing Summary:

---

Speed Grade: -3

Minimum period: 4.980ns (Maximum Frequency: 200.809MHz)

Minimum input arrival time before clock: 4.830ns

Maximum output required time after clock: 3.597ns

Maximum combinational path delay: No path found

La ruta crítica del algoritmo *Saturation* quedará determinada por el CASO1 y CASO2, que son los peores casos (los que poseen más cuentas). Al utilizar *pipeline* añadimos registros intermedios para que el cálculo total del algoritmo de *Saturation* fije como frecuencia óptima la frecuencia máxima de trabajo, 200Mhz.

## 3.6 Módulo Lightness.

Las ecuaciones utilizadas son las siguientes:

$$\text{Lightness} = 50 * (M + m)$$

Los módulos a diseñar son los siguientes. (Ver figura 5).

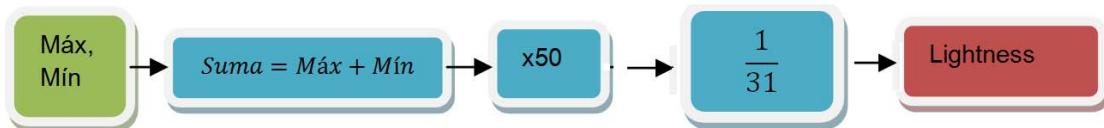
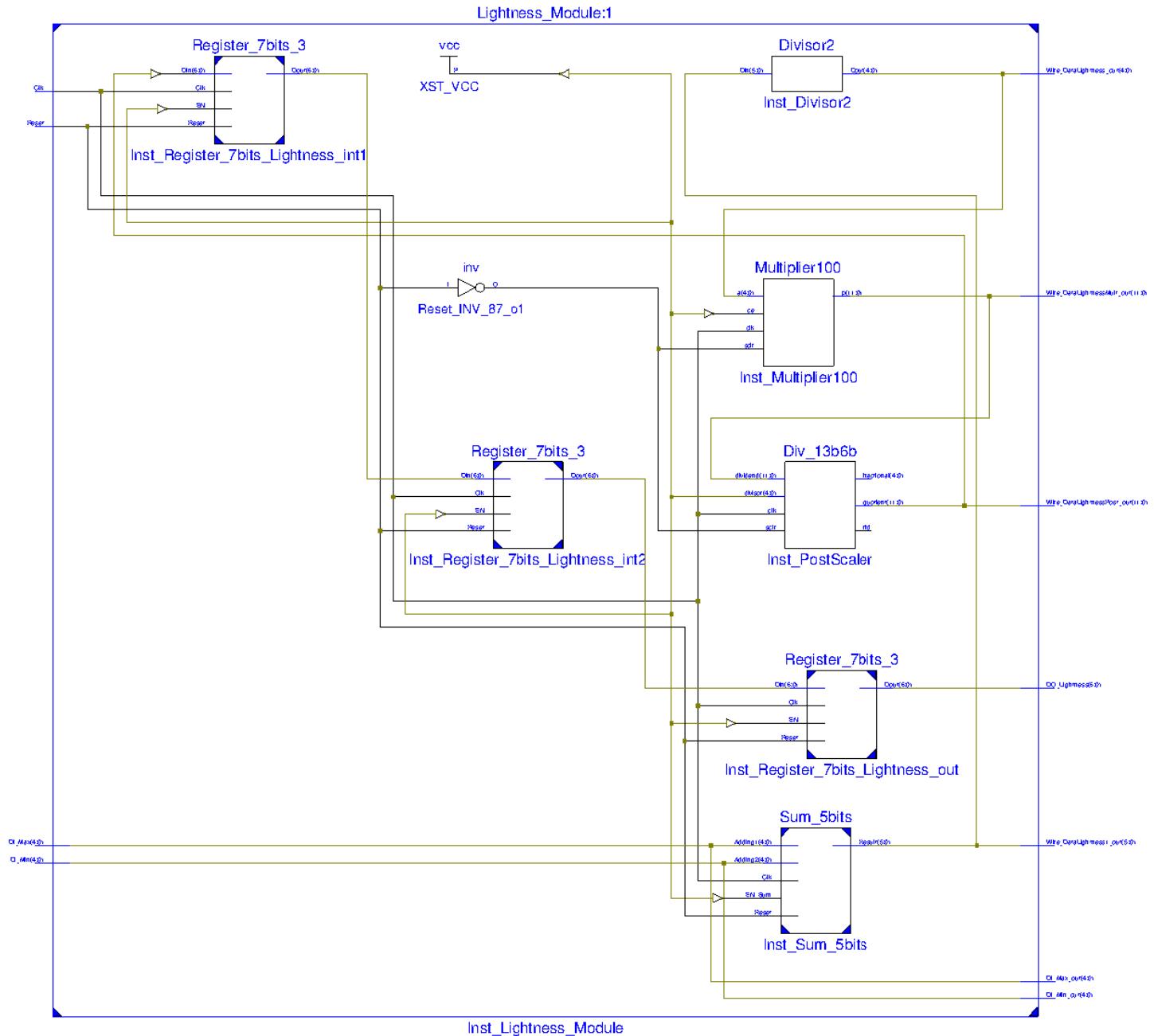


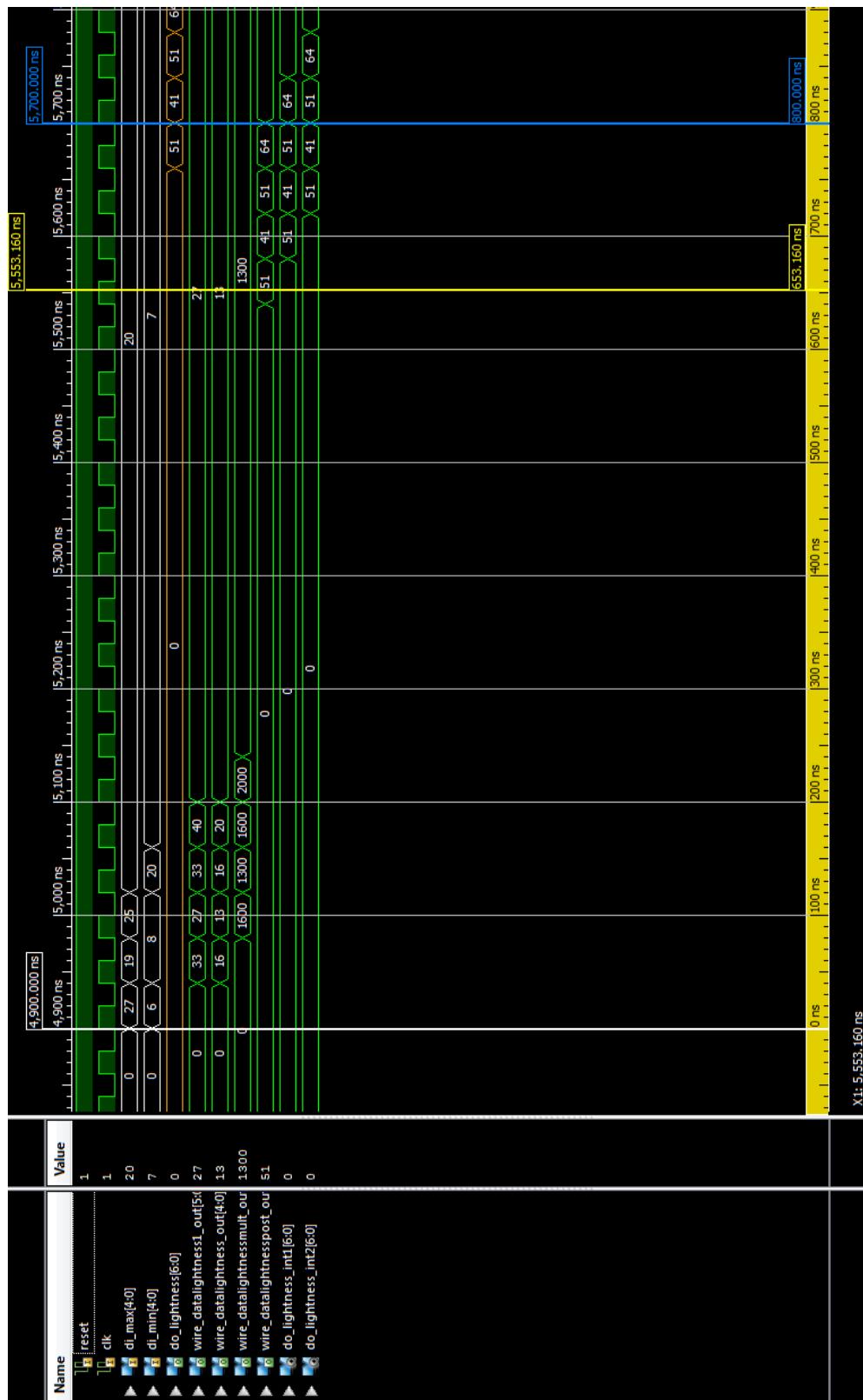
Figura 7: Algoritmo Lightness.

Se suma el valor del máximo y del mínimo del píxel recibido, y finalmente se multiplica por 50. Para conseguir un escalado correcto, el resultado final es dividido entre 31. Como los demás algoritmos, no debemos olvidar realizar primero la multiplicación y posteriormente la división.

### 3.6.1 Esquema RTL.



### 3.6.2 Simulación Post-Route



### 3.6.3 Recursos utilizados.

Para el implementar el algoritmo Lightness son necesarios los siguientes bloques de la FPGA.

1. 260 registros, de los cuales muchos de ellos son utilizados para la técnica *pipeline*.
2. 24 pines para entradas y salidas.
3. 1 DSP48A1s para el cálculo de operaciones de procesado de señal.
4. 172 Slices LUTs, para el cálculo de las divisiones.

### 3.6.4 Análisis de frecuencia.

Timing Summary:

-----

Speed Grade: -3

Minimum period: 4.980ns (Maximum Frequency: 200.809MHz)

Minimum input arrival time before clock: 4.512ns

Maximum output required time after clock: 4.414ns

Maximum combinational path delay: 4.507ns

La frecuencia máxima de trabajo será 200Mhz.

## 4. Algoritmo para la creación de un histograma HSL.

El módulo *HSL\_Histogram* está formado por dos submódulos *HSL\_HistogramReference* y *HSL\_HistogramSampled*, que almacenan el histograma de la imagen de referencia y el histograma de la imagen muestreada respectivamente.

El histograma se diseña como una memoria RAM cuyo funcionamiento es similar al de un contador. La memoria RAM implementada en la FPGA posee todas las entradas y salidas típicas: *ReadEnable*, *WriteEnable*, *Address*, *data\_in* y *data\_out*.

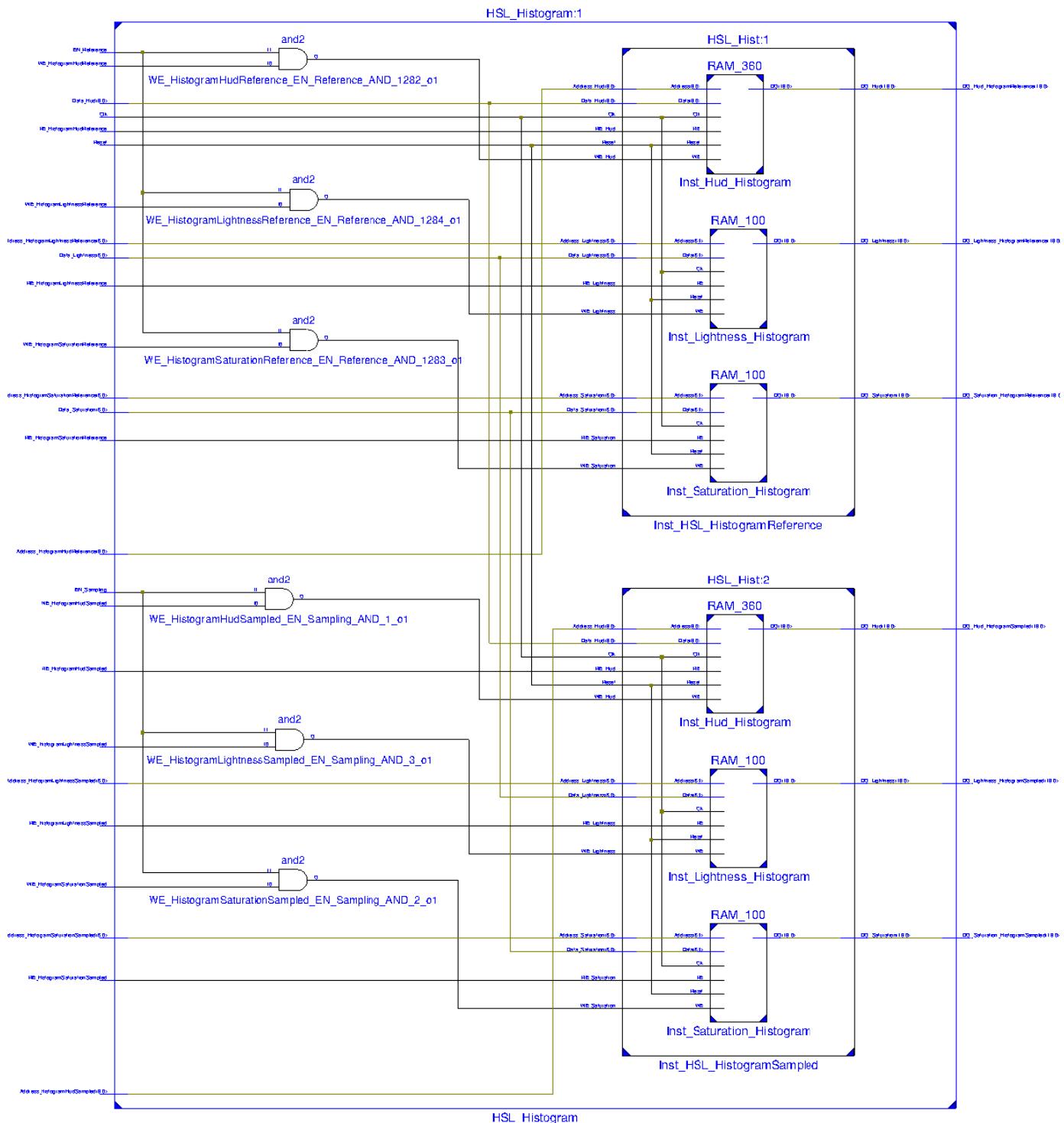
La creación del histograma es la siguiente:

5. Recibe el valor de un píxel en su entrada, *WE*=’1’.
6. Seleccionamos la posición de la memoria RAM equivalente al píxel.
7. Incrementamos el valor almacenado en dicha posición.
8. Repetimos el bucle hasta recibir la trama de píxeles completa.

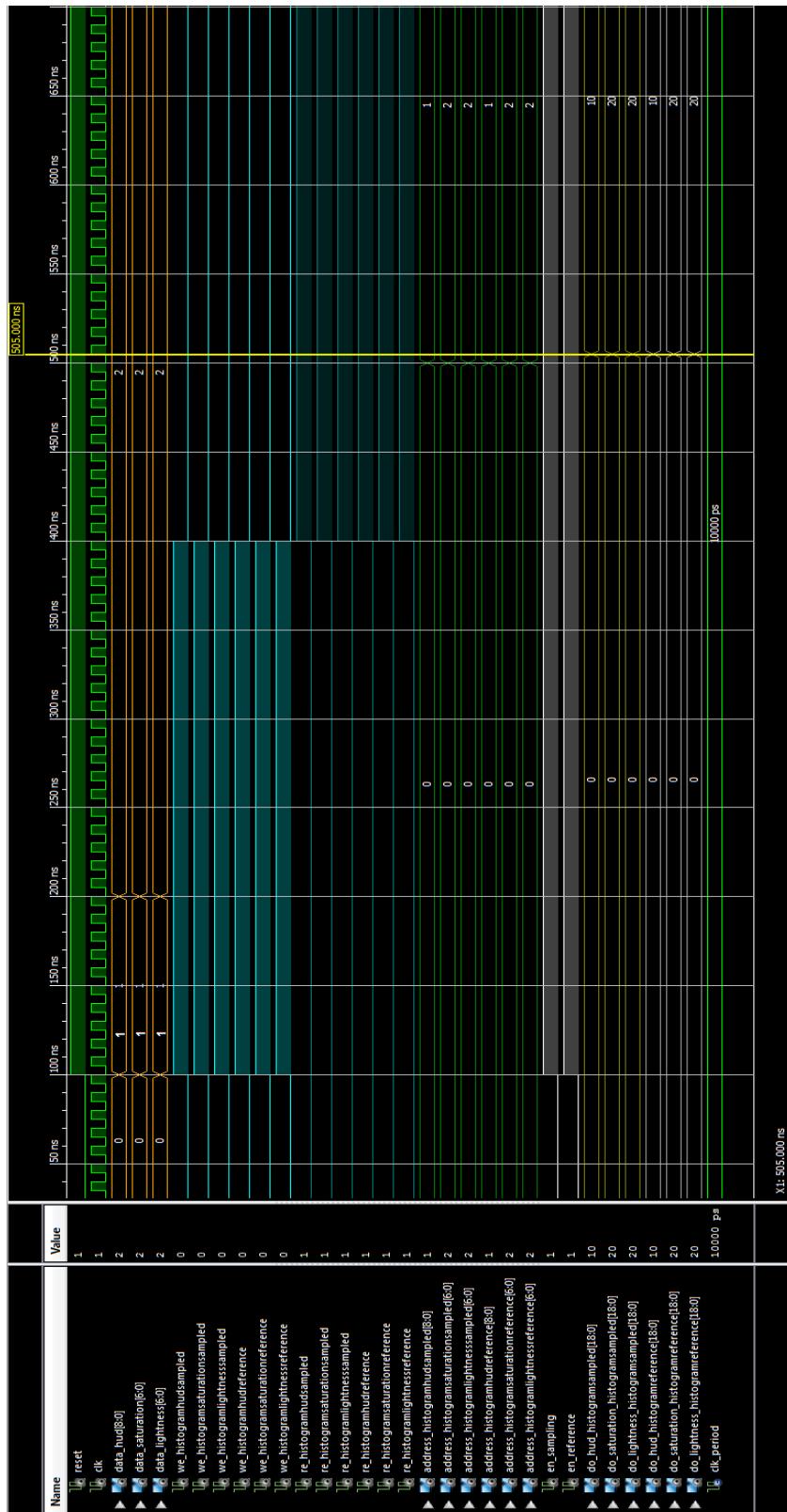
La lectura del histograma es la siguiente.

1. *Address* apunta a la primera posición de memoria de la RAM, *RE*=’1’.
2. Se extrae el valor de la posición de memoria a la salida *data\_out*.
3. Repetimos el bucle hasta recorrer todas las posiciones de memoria de la RAM.

## 4.1 Esquema RTL.



## 4.2 Simulación Post-Route



## 4.3 Recursos utilizados.

Para el implementar el histograma de referencia y el muestreado son necesarios los siguientes bloques de la FPGA.

1. 21508 registros.
2. 199 pines para entradas y salidas.
3. 15491 Slices LUTs, utilizados para la creación de las memorias RAM.
4. 6376+23745 Slices y LUT Flip Flop, para elaborar la lógica distribuida.

Una solución alternativa es utilizar una memoria RAM dedicada en configuración de doble puerto.

### a. Análisis de frecuencia.

Timing Summary:

-----

Speed Grade: -3

Minimum period: 6.166ns (Maximum Frequency: 162.189MHz)

Minimum input arrival time before clock: 9.563ns

Maximum output required time after clock: 3.597ns

Maximum combinational path delay: No path found

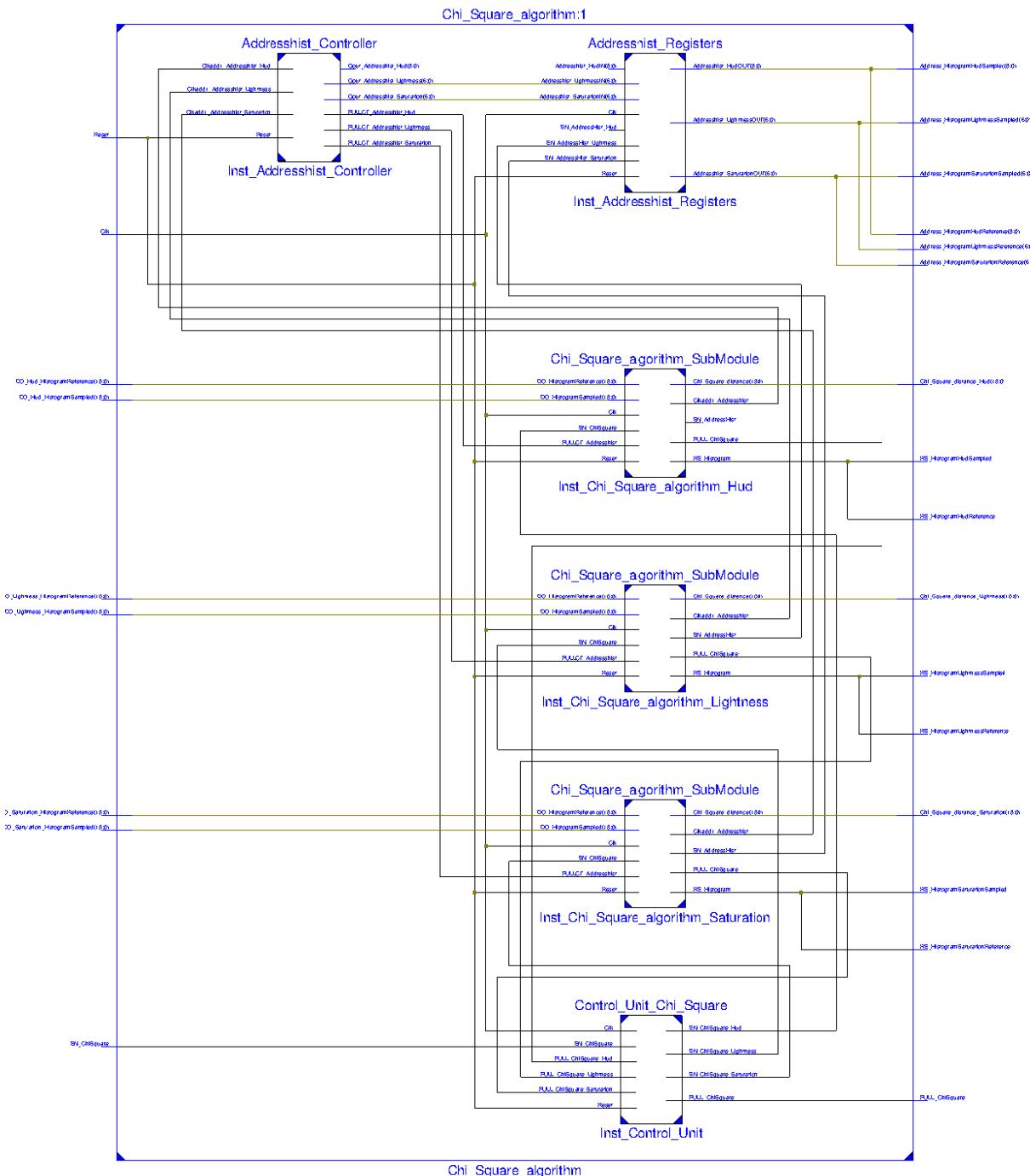
El tiempo consumido por el algoritmo para crear histogramas, no es relevante ya que su funcionamiento no supone un punto crítico en el sistema.

## 5. Cálculo de distancias entre histogramas, Chi-Cuadrado.

### 5.1 Esquema general.

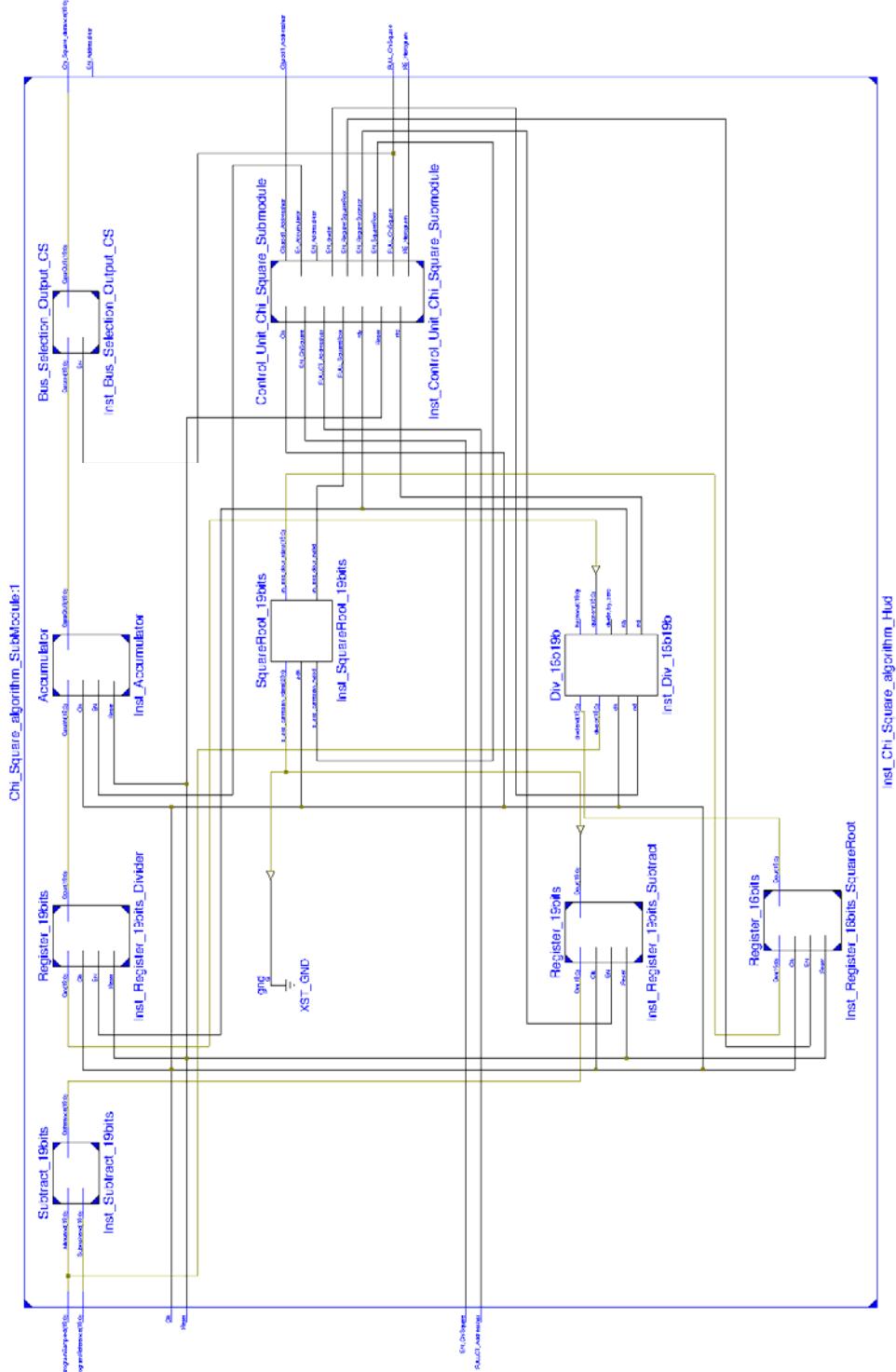
El diseño del módulo Chi-cuadrado está formado por los 3 submódulos conversores, uno por cada parámetro de HSL. Se añade un driver formador por (Addresshist\_Controller y Addresshist\_Register) que recorre el histograma de referencia y el muestreado.

## 5.1.1 Esquema RTL.

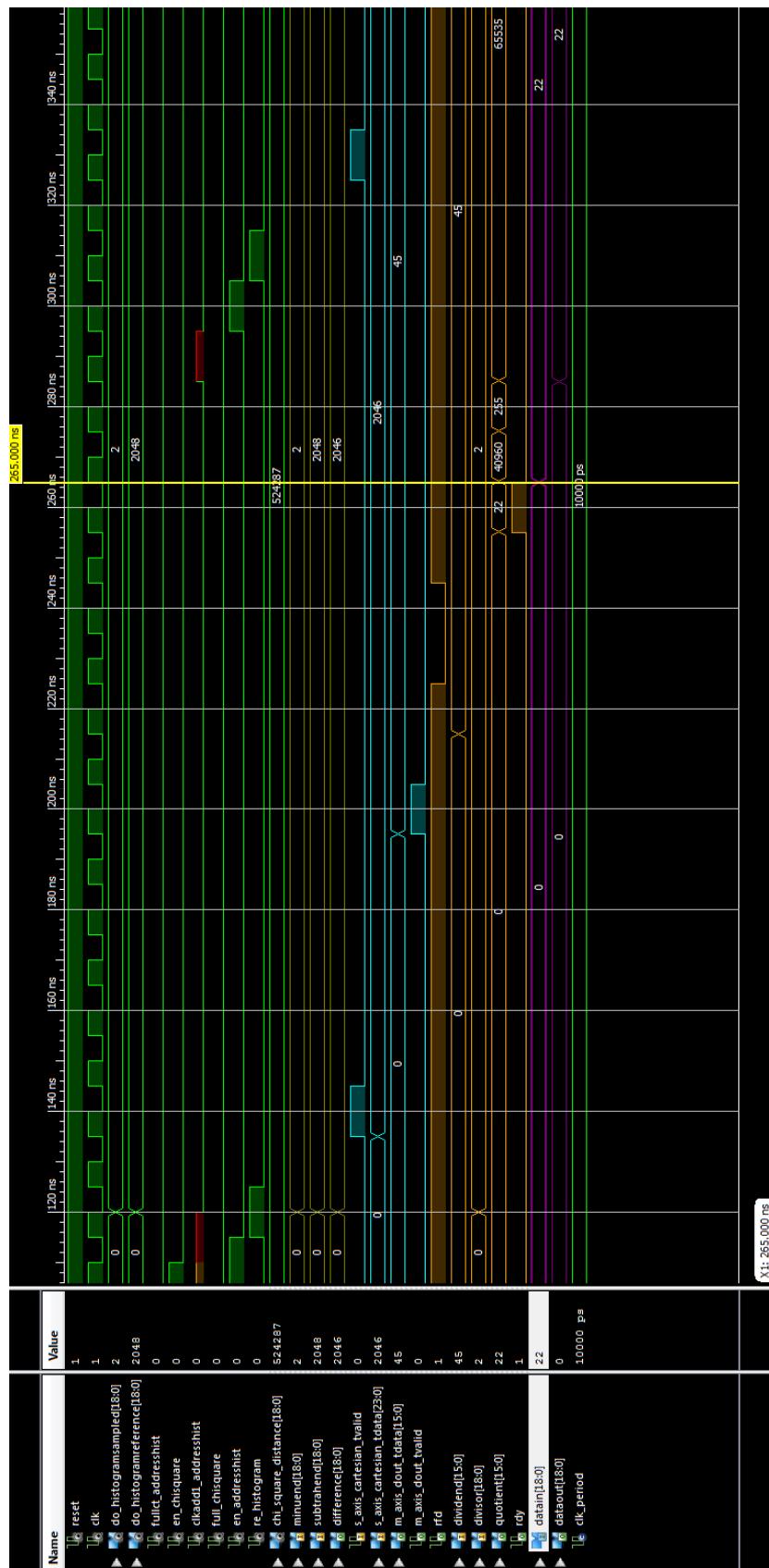


## 5.2 Submódulo Chi-cuadrado.

### 5.2.1 Esquema RTL.



### 5.2.2 Simulación Post-Route.



### 5.2.3 Recursos utilizados.

Para el implementar el comparador Chi-cuadrado son necesarios los siguientes bloques de la FPGA.

1. 309 registros, de los cuales muchos de ellos son utilizados para la técnica *pipeline*.
2. 65 pines para entradas y salidas.
3. 10 DSP48A1s para el cálculo de operaciones de procesado de señal.
4. 230+640 Slices y LUT Flip Flop, para elaborar la lógica distribuida.
5. 588 Slices LUTs, configurados como divisor.

La comparación entre histogramas es una carga computacional elevada para la FPGA. Los bloques implementados poseen cuentas complejas, divisiones y raíces cuadradas, que utilizan gran número de DSPs.

### 5.2.4 Análisis de frecuencia.

Timing Summary:

-----

Speed Grade: -3

Minimum period: 21.527ns (Maximum Frequency: 46.454MHz)

Minimum input arrival time before clock: 14.915ns

Maximum output required time after clock: 4.895ns

Maximum combinational path delay: No path found

Este proyecto no está pensado para un procesado en tiempo real, la comparación entre histogramas no tiene porqué ser un bloque optimizado temporalmente, como es el caso del convertidor RGB a HSL. Los histogramas de referencia y los histogramas de muestra son almacenados en memoria previamente, por lo tanto, el comparador Chi-cuadrado tiene todo el margen de tiempo que necesite.