DISEÑO DE UN SISTEMA EMBEBIDO EN UNA TARJETA DE DESARROLLO ALTERA DE2 PARA EL CONTEO DE INGRESO DE PERSONAS A TRAVÉS DE DETECCIÓN Y SEGUIMIENTO FACIAL

JAIME DARÍO JARAMILLO VILLOTA
JAVIER ALEXANDER POTOSÍ BENAVIDES

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA
SAN JUAN DE PASTO
2017

DISEÑO DE UN SISTEMA EMBEBIDO EN UNA TARJETA DE DESARROLLO ALTERA DE2 PARA EL CONTEO DE INGRESO DE PERSONAS A TRAVÉS DE DETECCIÓN Y SEGUIMIENTO FACIAL

JAIME DARÍO JARAMILLO VILLOTA
JAVIER ALEXANDER POTOSÍ BENAVIDES

Trabajo de grado para optar por el título de Ingeniero Electrónico

ASESOR:
ÁLVARO ANDRÉS JIMÉNEZ
INGENIERO ELECTRÓNICO

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA
SAN JUAN DE PASTO
2017

NOTA DE RESPONSABILIDAD

“La Universidad de Nariño no se hace responsable por las opiniones o resultados obtenidos en el presente trabajo y para su publicación priman las normas sobre el derecho de autor.”

Acuerdo 1. Artículo 324. Octubre 11 de 1966. emanado del honorable Consejo Directivo de la Universidad de Nariño.

NOTA DE ACEPTACIÓN

Firma del Jurado

Firma del Jurado

Pasto, 23 de mayo de 2017

AGRADECIMIENTOS

Ing. Álvaro Andrés Jiménez, asesor del presente trabajo de grado por su gestión y colaboración en el desarrollo del mismo

Programa de Ingeniería Electrónica de la Universidad de Nariño, los docentes, personal administrativo y de laboratorio por su contribución en nuestra formación académica y profesional.

DEDICATORIA

Dedico este trabajo a mi familia: Milton Jaime, Nancy, Camila y Sara, y todas las personas allegadas que de alguna u otra forma me inspiraron y creyeron en mí.

A Javier por haber hecho posible el éxito de este trabajo.

A Christian y Alexander por su colaboración y consejos durante el desarrollo de este trabajo.

Jaime Jaramillo

A mi papá, mamá y hermana, por su apoyo y comprensión.

A la familia Jaramillo Villota porque se convirtieron en mi segundo hogar, gracias por todo.

A Jaime, Christian y Alex, por su amistad y tenacidad.

A todos mis amigos, familiares y allegados que hicieron posible la consecución de esta meta.

Javier Potosi

RESUMEN

Las técnicas de análisis de imágenes y visión computacional han encontrado múltiples aplicaciones prácticas en campos como la industria, salud, educación, entretenimiento, sistemas de seguridad y vigilancia, entre muchos otros. Es por esto que se propone aplicar estas técnicas para el conteo de personas que ingresan a un lugar específico, como, por ejemplo, establecimientos públicos, de comercio, sistemas de transporte público, etc.

En este documento se describe el desarrollo de un sistema embebido en la tarjeta de desarrollo Altera DE2 con la FPGA Altera Cyclone II, que permite capturar digitalmente imágenes de una fuente de video analógica, con el fin de ser analizadas e implementar un algoritmo de detección y seguimiento facial para realizar así el conteo de personas.

Se realiza una revisión bibliográfica para establecer el flujo de trabajo adecuado para la creación de sistemas embebidos en la tarjeta de desarrollo, eligiendo herramientas que sean flexibles y permitan un rápido desarrollo del sistema. También se simula el algoritmo con el fin de establecer que partes se implementan en hardware.

Usando estos conocimientos se procede a crear una implementación en hardware del prototipo, creando un marco de trabajo y librerías necesarias para controlar y administrar los diferentes subsistemas y componentes de la tarjeta de desarrollo.

ABSTRACT

Computer vision and image analysis techniques have found many practical applications in fields like industry, healthcare, education, entertainment, security systems and surveillance among many others. For this reason, it is proposed to apply these techniques to count people entering a specific place, i.e. public spaces, businesses, mass transportation systems, etc.

This document describes the development of an embedded system in the Altera DE2 development board using the Altera DE2 FPGA, that allows to digitally capture images from an analog video source to analyze them and implement a facial detection and tracking algorithm, thus achieving a person counting system.

A bibliographic review is made to establish the adequate workflow for the development of embedded systems in the Altera DE2 board, using tools that place emphasis in flexibility and rapid development. The algorithm is also simulated to define which parts are going to be implemented in hardware.

Using this knowledge, the hardware implementation of the prototype is developed, creating the framework and the essential libraries and drivers required to manage the different subsystems and devices of the development board.

CONTENIDO

LISTA DE TABLAS

pág.

LISTA DE FIGURAS

pág.

GLOSARIO

CMY: del inglés *Cyan-Magenta-Yellow,* Cian-Magenta-Amarillo. Espacio de color basado en los colores secundarios, pues se restan al blanco puro para obtener los colores requeridos.

DMA: del inglés *Direct Memory Access*, acceso directo a memoria. Es la capacidad que permite a ciertos elementos acceder a la memoria central del sistema.

FPGA: del inglés *Field Programmable Gate Array,* arreglo de compuertas programables en campo. Es un dispositivo lógico que puede ser programado después de ser fabricado. Contiene elementos lógicos programables e interconexiones reconfigurables que permiten conectar físicamente a los diferentes elementos lógicos.

FPS: del inglés *Frames per Second,* es la frecuencia a la cual una pantalla muestra imágenes consecutivas llamadas cuadros.

Frame: del inglés *frame*, cuadro. Es una muestra de espacio de tiempo de una secuencia de video la cual está compuesta de *scanlines* o líneas de escaneo.

Histograma bimodal: es un histograma con dos picos de datos notorios, también llamados modos.

Histograma: es un tipo de gráfico que provee una interpretación visual de datos numéricos, indicando el número de datos que se encuentran dentro de un rango de valores.

HSV: del inglés *Hue-Saturation-Value,* Matiz o Color-Saturación-Valor. Espacio de color perceptual, basado en el sistema de visión humano. Ampliamente conocida y utilizada para el análisis de imágenes.

Interpolación proximal o del vecino más cercano: Es una forma de aumentar el tamaño de imagen y consiste en clonar los pixeles del mismo color, la imagen resultante es más grande que la original.

NTSC: del inglés *National Television System Committee,* Comité Nacional de Sistema de Televisión.

Orden de escaneo Raster: forma de escaneo de imagen, en la cual el elemento recorre la imagen una línea de escaneo horizontal a la vez, iniciando en la parte superior izquierda hasta finalizar en la parte inferior derecha de la imagen.

RAM: Del inglés *Random-access Memory,* memoria de acceso aleatorio. Dispositivo de almacenamiento volátil, a la cual se puede acceder de manera

aleatoria pues se puede escribir o leer con un tiempo similar para cualquier dirección del dispositivo.

RCA: Conectores de tipo eléctrico. Se utilizan como medio de transmisión para señales de audio y video.

RGB: del inglés *Red-Green-Blue,* rojo-verde-azul. Espacio de color que está basado en los colores primarios, los cuales al ser combinados se utilizan para representar imágenes.

Scanline: del inglés *Scanline*, Líneas de escaneo. Es una fila o línea dentro del patrón de escaneo raster, como el utilizado en los monitores.

SoC: del inglés *System on a chip*, dispositivo que integra los elementos necesarios para crear un sistema.

SoPC: del inglés *System on a Programable Chip*, hace referencia a la combinación de un procesador con un elemento hardware programable, incorpora un sistema configurable dentro de un chip.

SRAM: Del inglés *Static Random Access Memory,* memoria estática de acceso aleatorio. Es un tipo de memoria RAM volátil, que almacena información de forma estática. A diferencia de las RAM dinámicas, es memoria SRAM no requiere circuito de refresco.

YCbCr: espacio de color perceptual cuyas siglas representan: Y-luma, Cb componente de color azul, Cr componente de color verde. Se utiliza para la transmisión, almacenamiento y compresión de imagen y video.

GLOSARIO DE HERRAMIENTAS DE SOFTWARE

Diseño Bottom-Up: diseño de abajo arriba. Metodología que se enfoca en el diseño de sistemas más pequeños y ágiles mediante lenguajes de programación relativamente cortos que no han sido divididos en muchos componentes. Posteriormente ensambla los pequeños sistemas hasta formar sistemas más complejos.

Diseño Top-Down: diseño de arriba abajo. Es una metodología de diseño que parte de lo general a lo particular, un bosquejo general del sistema se refina cada vez con mayor detalle hasta llegar al nivel de implementación, cada componente es una caja negra que tiene un bosquejo de sus funciones más no de su implementación, la cual se detalla una vez los componentes del nivel superior han sido definidos.

DSP: del inglés *Digital Signal Processing*, procesamiento digital de señales. Hace referencia a las diferentes técnicas que se utilizan para mejorar la exactitud y confiabilidad de las señales de información.

HDL: del inglés *Hardware description language*, lenguaje de descripción de hardware. Es un lenguaje de programación especializado que se utiliza para describir la estructura, operaciones y diseño de circuitos electrónicos y circuitos digitales.

LPM: del inglés Library of Parameterized Modules

Qsys: Qsys es una herramienta de integración incluida en el paquete Quartus II™. Qsys captura diseños de hardware para sistemas embebidos a un nivel de abstracción alto y automatiza la tarea de definir e integrar diferentes componentes HDL personalizados.

Quartus II: es la herramienta de software que provee Altera en un ambiente de diseño multiplataforma que se adapta fácilmente a los requerimientos del diseño, además de ser ideal para el diseño de SoPC.

VHDL: acrónimo de VHSIC y HDL, del inglés *very high speed integrated circuit* y *hardware description language* respectivamente. Lenguaje de descripción de hardware diseñado y optimizado para describir el comportamiento de circuitos y sistemas digitales.

INTRODUCCIÓN

El presente trabajo de investigación hace uso de las herramientas provistas para el estudio de la visión computacional, así como de los dispositivos de lógica programable, con el objeto de diseñar un sistema embebido en una tarjeta de desarrollo Altera DE2 para realizar el conteo de personas que ingresan a un determinado sitio a través de la detección y seguimiento facial. Se pretende explorar nuevos temas en el tratamiento de imágenes y video, y especialmente su implementación en FPGAs, pues se encuentra que el uso e investigación de estos dispositivos en el Departamento de Ingeniería Electrónica de la Universidad de Nariño se puede explorar aún más a fondo.

Inicialmente se realiza un estudio del tratamiento de imágenes y video enfocado a la detección y seguimiento facial. Posteriormente se realiza una investigación de la tarjeta de desarrollo Altera DE2, la cual contiene la FPGA Cyclone II con el propósito de estudiar sus características y limitaciones para finalmente implementar los métodos estudiados previamente haciendo uso de las herramientas de hardware y software que ofrece el dispositivo.

A través de este proyecto se pretende desarrollar un sistema de bajo costo para la Universidad, para el conteo de ingreso de personas con la finalidad de que los establecimientos logren utilizar está información para desarrollar estrategias que mejoren su interacción con los clientes. En el ámbito académico, se espera motivar a los estudiantes a la investigación y uso de nuevas tecnologías, así como desarrollar sus capacidades para determinar la idoneidad de cada herramienta para la implementación de sus aplicaciones.

OBJETIVOS

OBJETIVO GENERAL

Desarrollar un prototipo que permita la captura de imágenes provenientes de una fuente analógica para la detección y seguimiento facial en condiciones apropiadas de luz, aplicado al conteo del ingreso de personas a un establecimiento, en la tarjeta de desarrollo Altera DE2.

OBJETIVOS ESPECÍFICOS

- Realizar la revisión bibliográfica de métodos y algoritmos de visión computacional enfocados en los métodos de detección y seguimiento facial.

- Implementar los métodos estudiados y simulados en el dispositivo de lógica reconfigurable FPGA Cyclone II.

- Desarrollar librerías y software controlador que permita aprovechar los periféricos de entrada de video analógico en la tarjeta de desarrollo de la FPGA Altera DE2.

- Diseñar un prototipo funcional implementado en hardware que permita el conteo de personas.

1. MARCO TEÓRICO

1.1. VISIÓN COMPUTACIONAL

La visión computacional es el análisis automático de imágenes y videos a través de una computadora con el objetivo de comprender un poco más el mundo. La visión computacional está inspirada en la capacidad del sistema de visión humana; inicialmente se creía de manera errónea que sería un problema relativamente fácil de resolver, debido a que nuestro sistema de visión hace que la tarea parezca intuitiva para nuestro cerebro[1].

En la actualidad la mayor parte de la información está diseñada para un consumo visual: gráficos, imágenes, video, o en general contenido multimedia. La velocidad con la que este tipo de información se transmite, almacena, procesa y muestra en formato digital está incrementando rápidamente, y así los métodos para la eficiencia de transmisión y para mantener e incluso mejorar la calidad visual de esta información son temas de interés para la ingeniería.

Existen diversas aplicaciones que usan el procesamiento de imágenes para su funcionamiento. Virtualmente, todas las ramas de la ciencia tienen una disciplina que utiliza dispositivos de registro o diferentes tipos de sensores que recolectan información de imágenes de nuestro entorno. Estos datos a menudo son multidimensionales y pueden ser adaptados a un formato entendible para la visión humana[2].

Una característica importante de las imágenes y el video digital es que son señales multidimensionales. En el estudio del procesamiento de señales digitales, estas usualmente son funciones unidimensionales que dependen del tiempo. Sin embargo, las imágenes son funciones de dos e incluso tres dimensiones espaciales, mientras que el video digital consta de tres dimensiones espaciales y una de tiempo[3]. En conclusión, la dimensión de una imagen es el número de coordenadas que son requeridas para ubicar un punto dentro de esta, es por esto que el procesamiento de imágenes digitales y especialmente de video es tan complejo, requiriendo gran cantidad de recursos de hardware y software[4].

---

[1] DAWSON-HOWE, Kenneth. A Practical Introduction to Computer Vision With OpenCV. Wiley, 2014. ISBN: 978-11-1884-845-6.
[2] BOVIK, Alan Conrad. Handbook of Image and Video Processing. Academic Press, 2005. ISBN: 978-01-2119-792-6.
[3] Ibíd, SOLOMON, Chris y Toby BRECKON. Fundamentals of Digital Image Processing. Wiley, 2005. ISBN: 978-04-7084-472-4.
[4] BOVIK. Op. cit.

1.1.1. Definición de imagen.

Una imagen es una proyección de dos dimensiones de una escena tridimensional capturada por un sensor, siendo una función continua de dos coordenadas en el plano[5]. Una vez la imagen continua es muestreada y cuantizada se convierte en una imagen digital, la cual se puede considerar como una representación discreta de datos, provista de información espacial (disposición) e intensidad (color). Una imagen también puede ser tratada como una señal multidimensional[6].

1.1.2. Digitalización de imágenes.

Antes de ser capturadas, las señales e imágenes que se encuentran en nuestro entorno naturalmente son análogas. Por análogo podemos entender que la señal existe en un dominio continuo espacio-tiempo y que toma valores que provienen de un plano infinito de posibilidades. Por digital se entiende que la señal está definida dentro de un dominio discreto espacio-tiempo y que toma valores de un conjunto discreto de posibilidades. Antes de iniciar con el procesamiento digital, se debe convertir la señal análoga a digital, para lo cual se hace necesario dos procesos diferentes: muestreo y cuantización.

1.2. IMÁGENES A COLOR

En el sistema visión humano los colores son captados como combinaciones de longitudes de onda largas, medias y cortas; las cuales corresponden aproximadamente a los tres colores primarios que se usan en los sistemas estándar de las cámaras de video: rojo (*Red*-R), verde (*Green*-G) y azul (*Blue*-B). Como se mencionó anteriormente, el sistema RGB es usado por cámaras y sistemas de visualización de video, mientras que el sistema YUV se usa en el procesamiento de imágenes, así como en la transmisión de televisión.

Una imagen contiene uno o más canales de color que definen la intensidad o el color de un pixel en particular. Cada pixel únicamente tiene un valor numérico que representa el nivel de la señal en ese punto de la imagen. La conversión de este conjunto de números a una imagen se logra a través de un mapa de color, el cual asigna un tono a cada nivel numérico y da una representación visual de los datos. El mapa de color más común es la escala de grises, la cual asigna todos los tonos de gris, desde negro hasta blanco de acuerdo al nivel de la señal[7].

Para comprender de manera más precisa la teoría y tratamiento de las imágenes a color, es necesario poseer conceptos claros sobre luminancia, luma, crominancia y

---

[5] DAWSON-HOWE. Op. cit.
[6] SOLOMON y BRECKON. Op. cit.
[7] BOVIK. Op. cit.

croma. Frecuentemente se llama de manera equivocada luma a la luminancia y viceversa, pues aparentemente ambos conceptos se tratan de un solo concepto. En términos generales, la luminancia hace referencia a la cantidad de luz o brillo ponderada con las curvas de sensibilidad a los colores de la visión humana. El término luma describe la suma ponderada de las componentes RGB no lineales, cuyo resultado representa la luminosidad[8]. Por otra parte, dado que la visión humana es menos sensible a los cambios de color que a los cambios de luz, se aprovecha esta característica para reducir la información de color de una imagen y mantener la luma con el mayor detalle posible. Una vez se obtiene la componente luma, se forman dos señales de diferencia de color (crominancia): azul menos luma y rojo menos luma. Finalmente, el submuestreo reduce el detalle en las componentes de diferencia de color (croma)[9].

A diferencia de las imágenes en escala de grises, donde tenemos un único valor para cada pixel que representa la luma, las imágenes a color están representadas por la luma y croma, donde todo el espectro de colores se puede representar como un triple vector[10].

La mayoría de las teorías y algoritmos para el procesamiento de imagen y video digital han sido desarrollados para valores únicos, monocromáticos (escala de grises) o imágenes de poca intensidad, mientras que las imágenes a color son señales vectoriales. Sin embargo, estas técnicas son aplicadas frecuentemente en las imágenes a color a cada componente de color por separado como si fuese una imagen independiente para ser procesadas y posteriormente recombinar los resultados.

La crominancia generalmente está asociada en menor medida con los detalles de la imagen o cambios significativos de luz. El ojo humano posee un mayor ancho de banda para la percepción de luminancia que para la crominancia, característica que es ampliamente utilizada por algoritmos de compresión que usan representaciones de color alternativas, como es el caso de YUV y almacenan, transmiten o procesan las componentes de crominancia utilizando un ancho de banda más pequeño (menor cantidad de bits) que la componente de luma[11].

Generalmente en el procesamiento de imágenes, cada color está representado por 8 bits (1 byte) lo que significa que se pueden representar 256 tonos diferentes

---

[8] POYNTON, Charles. Digital Video and HD: Algorithms and Interfaces. Morgan Kaufmann, 2003. ISBN: 978-15-5860-792-7.
[9] Ibíd.
[10] DAWSON-HOWE. Op. cit, SOLOMON y BRECKON. Op. cit.
[11] BOVIK. Op. cit.

para cada canal. Combinando los valores de los tres canales, cada pixel puede representar 16.777.216 colores diferentes[12].

1.3. ESPACIOS DE COLOR

Es la representación que se usa para almacenar los colores, específicamente la cantidad y naturaleza de los canales de color.

Desde el punto de vista matemático, una imagen no es más que un conjunto organizado espacialmente en el que cada pixel tiene una coordenada. La escala de gris o las imágenes binarias son matrices de dos dimensiones que asignan un valor numérico a cada pixel el cual representa la intensidad en ese punto. En contraste, las imágenes a color son matrices de tres dimensiones que asignan tres valores numéricos a cada pixel, para el caso del espacio de color RGB cada valor corresponde a rojo, verde y azul respectivamente[13].

Uno de los espacios de color más utilizado en el procesamiento de imágenes es el YCbCr, pues este asemeja más el comportamiento de la visión humana dando prioridad a la información de luminosidad a la cual el ojo humano es más sensible; además de comprimir las componentes de color Cb y Cr, lo que se ve reflejado en una reducción de la cantidad de información a procesar. Adicionalmente, el espacio de color YCbCr se utiliza para la transmisión, almacenamiento y compresión de imagen y video, como por ejemplo los estándares JPEG y MPEG respectivamente.

1.4. SEGMENTACIÓN DE IMÁGENES

La segmentación es la tarea de encontrar grupos de pixeles que “van juntos”. Estadísticamente este problema se conoce como análisis de grupos. En el campo de la visión computacional, la segmentación de una imagen es uno de los problemas más antiguos y ampliamente estudiados[14].

Para varias aplicaciones, como en la detección de piel humana, la segmentación se encarga de separar el fondo y demás elementos que no son necesarios en la imagen del objeto que es primordial para el desarrollo de la aplicación.

---

[12] MOESLUND, Thomas B. Introduction to Video and Image Processing: Building Real Systems and Applications. Springer, 2012. ISBN: 978-14-4712-502-0.
[13] SOLOMON y BRECKON. Op. cit.
[14] SZELISKI, Richard. Computer Vision: Algorithms and Applications. Springer, 2010. ISBN: 978-18-4882-934-3.

Para el desarrollo del proyecto, se requiere separa los pixeles de una imagen que representen un rostro mediante la detección de piel humana del resto de la imagen, esto mediante el proceso de *Thresholding* y la crominancia de la imagen.

1.4.1. Imágenes binarias

Una imagen binaria es aquella en la cual existe un único valor por pixel, por ejemplo, blanco o negro[15]. Generalmente, una imagen binaria se obtiene de extraer la información esencial de una imagen y se usa para la localización de objetos, detección de bordes, o descubrir la presencia o ausencia de alguna propiedad de la imagen[16].

A diferencia de las imágenes en escala de grises, en una imagen binaria solo un bit es asignado para cada pixel, lo que significa que existen dos valores dentro de la escala de gris, 0 y 1. Estos valores son interpretados como números Booleanos[17].

Aplicadas al diseño y desarrollo del proyecto, las imágenes binarias se obtienen después del paso de segmentación o *thresholding* en donde el valor 1 represente la existencia de rostro o piel humana, y el 0 la ausencia del mismo. Se debe considerar que para conseguir una imagen binaria con resultados óptimos, esta se debe someter a un proceso de filtrado, pues el ruido está presente durante el proceso incluso desde el momento de capturar la señal de entrada, la cual se realiza por medio de una cámara de video análoga.

1.5. VIDEO DIGITAL Y ANÁLOGO

El video se puede considerar como una secuencia de imágenes, por tanto, los métodos utilizados para el procesamiento de imágenes se pueden aplicar aquí, con la diferencia que se procesa una imagen a la vez[18].

Para el caso en concreto, se requiere aprovechar la entrada análoga de video que posee la tarjeta de desarrollo Altera DE2, así como el decodificador de video ADV7180 integrado en el mismo dispositivo.

1.6. PROCESAMIENTO DE VIDEO

Además de la detección el proyecto considera el seguimiento facial, es decir la detección, ubicación y rastreo del objeto en cada cuadro de video; por tanto es

---

[15] DAWSON-HOWE. Op. cit.
[16] BOVIK. Op. cit.
[17] Ibíd.
[18] MOESLUND. Op. cit.

necesario aplicar métodos similares al procesamiento de imágenes, eliminando elementos estáticos en la secuencia además de los que no se detecten como piel humana.

La detección de objetos en movimiento se usa en secuencias que contiene objetos en movimiento que pueden ser autos, personas, etc. El movimiento relativo de objetos en una escena hace posible segmentar estos objetos del resto de la escena[19].

El algoritmo básico para detectar objetos en movimiento en una secuencia de video consiste de cinco pasos los cuales se describen a continuación y se aprecian en la Figura 1:

1. Establecer y guardar la imagen de referencia.
2. Capturar la imagen actual.
3. Realizar la diferencia o substracción de las imágenes.
4. Aplicar el proceso de *thresholding* a la imagen resultante.
5. Filtrar la imagen final[20].

[19] DAWSON-HOWE. Op. cit.
[20] MOESLUND. Op. cit.

Figura 1.Detección de objetos en movimiento

Fuente: MOESLUND, Thomas B. Introduction to Video and Image Processing: Building Real Systems and Applications. Springer (2012).

Si calculamos la diferencia entre el cuadro actual de una secuencia de video y la imagen base la cual no presenta movimiento, entonces podemos identificar los objetos que presentan movimiento en la escena. Cabe aclarar que para la imagen resultante se pueden presentar falsos positivos (pixeles que realmente no están en movimiento) o falsos negativos (pixeles que no se encuentran estáticos). El ruido es una de las causas principales para que el proceso no sea correcto, ocasionando que sea muy difícil minimizar estos factores. Un mal proceso de *thresholding* puede causar que exista demasiado ruido en la imagen (para valores de *T* muy pequeños), o que los objetos en movimiento no sean detectados (si el valor de *T* es muy alto)[21].

Los pixeles fuera de la silueta del objeto se pueden remover utilizando un filtro de mediana (el cual sustituye el valor del pixel por la mediana de los pixeles vecinos) o la operación morfológica de apertura; y los vacíos dentro del mismo se logran solventar mediante la operación morfológica de cierre.

1.7. DETECCIÓN DE ROSTROS

La detección de rostros ha sido un campo de investigación muy activo dentro de la visión computacional, pues cada día son más los sistemas que utilizan diversos algoritmos dedicados a esta función con el propósito de mejorar su desempeño.

[21] DAWSON-HOWE. Op. cit.

Por ejemplo, las cámaras digitales o de teléfonos inteligentes utilizan la detección de rostros para mejorar el auto enfoque y dentro de los sistemas de video inteligente para realizar el seguimiento de un individuo[22].

Una definición conceptual de detección facial es: Dada una imagen arbitraria, el objetivo de la detección facial es determinar si hay rostros presentes en una imagen, y si es así, determinar la localización y los límites de estos[23].

1.7.1. Detección de piel

La detección de color de piel humana ha probado ser un método robusto y bastante efectivo para la identificación facial, a pesar de que las personas presentan diferencias en su color de piel, varios estudios han mostrado que el sistema de visión humano es más susceptible a los cambios de intensidad de luz que a los cambios de color[24]. Por lo anterior, se pueden utilizar para la detección de piel diferentes espacios de color que aprovechen esta característica, en particular YCrCb, el espacio de color usado para fuentes NTSC.

El color de piel permite un procesamiento rápido y bastante resistente ante las variaciones geométricas del patrón facial. La experiencia sugiere que la piel humana tiene un color característico fácilmente reconocible por la visión humana, por lo cual es una técnica que vale la pena aplicar y simplificaría su detección[25], ya que los demás métodos al requerir de la imagen en su totalidad, no son adecuados para ser implementados en sistemas embebidos en tiempo real.

El color de piel no es una propiedad física sino perceptual, es por esto que un modelado que se aproxime a las características de la visión humana es deseable para la detección. Una de las principales consideraciones para la detección facial a través del color de piel es la elección del espacio de color. Existen diversos estándares con origen en la colorimetría, gráficos computacionales y transmisión

---

[22] SZELISKI. Op. cit.

[23] YANG, Ming Hsuan, et al. Detecting faces in images: A survey. En: IEEE Transactions on Pattern Analysis and Machine Intelligence. 2002. vol. 24

[24] GRAF, Hans Peter, et al. Locating faces and facial parts. En: Proceedings of the First International Workshop on Automatic Face and Gesture Recognition. 1995, GRAF, Hans Peter, et al. Multi-modal system for locating heads and faces. En: Proceedings of the Second International Conference on Automatic Face and Gesture Recognition, 1996 (Killington). IEEE, 1996

[25] BERBAR, Mohamed A., et al. Faces and Facial Features Detection in Color Images. En: Geometric Modeling and Imaging – New Trends (GMAI06) (Washington). IEEE, 2006

de señales de video. Gran parte de estos han sido aplicados al problema del modelado del color de piel[26].

El espacio de color YCrCb se utiliza para el desarrollo del prototipo, pues este es la salida del decodificador ADV7180 de la tarjeta de desarrollo. Adicionalmente se aprovecha su reducción en las componentes de color y su aprovechamiento de la componente de intensidad para la representación de imágenes, hecho que se asemeja las características de la visión humana[27].

1.8. SISTEMAS EMBEBIDOS

Los sistemas embebidos son un tipo de sistemas computacionales diseñados para realizar tareas específicas, en contraste a los sistemas de propósito general como los computadores personales, los cuales están diseñados para ser flexibles y soportar una gran variedad de usos y aplicaciones. Estas aplicaciones se desarrollan teniendo en cuenta la cantidad de recursos disponibles en el sistema.

Un sistema embebido se dedica a un número limitado de tareas con un alcance especifico, por esto, su diseño puede ser optimizado para reducir su costo. Un buen diseño solo debe contener los suficientes recursos de hardware para alcanzar las funcionalidades requeridas por la aplicación. Un sistema de propósito general, por el contrario, debe ser flexible y soportar una variedad de escenarios, por esta razón las aplicaciones desarrolladas cuentan con recursos de hardware relativamente abundantes. Desde esta perspectiva, un sistema embebido se puede entender como un sistema computacional con recursos de hardware limitados[28].

1.8.1. FPGA

Las FPGA (Field Programmable Gate Array o arreglo de compuertas programables en campo) son circuitos integrados que contienen celdas lógicas programables junto con conexiones configurables entre estos bloques. Los ingenieros diseñadores pueden configurar estos dispositivos para realizar una gran variedad

---

[26] VEZHNEVETS, Vladimir, et al. A Survey on Pixel-Based Skin Color Detection Techniques. En: Proceedings of GraphiCon 2003. 2003. vol. 85

[27] KIM, Woo-Shik Kim; Dae-Sung Cho; Hyun Mun. Interplane prediction for RGB video coding. En: Image Processing, 2004. ICIP '04. 2004 International Conference on (Singapore, Singapore). IEEE, 2004

[28] CHU, Pong P. FPGA Prototyping by Verilog Examples: Xilinx Spartan-3 Version. John Wiley & Sons, 2008. ISBN: 978-04-7018-532-2.

de tareas. Un elemento se implementa especificando la función de cada celda lógica y seleccionando las conexiones de cada interruptor programable[29].

1.8.2. Tarjeta de desarrollo Altera DE2.

La tarjeta de desarrollo Altera DE2 del año 2007 es el vehículo ideal para el diseño de prototipos con características avanzadas y capacidades de procesamiento en tiempo real. Las herramientas CAD provistas exponen un amplio rango de características, desde circuitos simples hasta varios proyectos multimedia.

La tarjeta cuenta con el chip Cyclone II EP2C35F672C6N que dispone de 475 pines, 33.216 elementos lógicos complejos (LEs), 483.840 bits de RAM (organizados en 105 bloques de 4 Kbits), 35 multiplicadores y 4 PLLs (utilizados para generar señales de reloj).

La tarjeta cuenta con una serie de componentes conectados a pines determinados del dispositivo, los cuales son programables. Las entradas y salidas digitales con las que cuenta la tarjeta de desarrollo son:

- 4 *pushbuttons* (KEY0-KEY3) de entrada, se encuentran normalmente en alto, se genera un pulso bajo al ser presionados. Tienen implementados circuitos que eliminan el rebote.

- 18 interruptores o *switches* digitales (SW0-SW17) de entrada. No poseen circuitos eliminadores de rebote.

- Circuitos generadores de señales de reloj a 27 MHz, 50 MHz y un conector para reloj externo.

- 9 LEDs (LEDG0-LEDG8) verdes.

- 18 LEDs (LEDR0-LEDR17) rojos.

- 8 *displays* de 7 segmentos de salida. Los LEDs del *display* se encienden con una señal en bajo y se apagan en alto.

- 1 *display* LCD de 16×2 (2 filas de 16 caracteres) de salida.

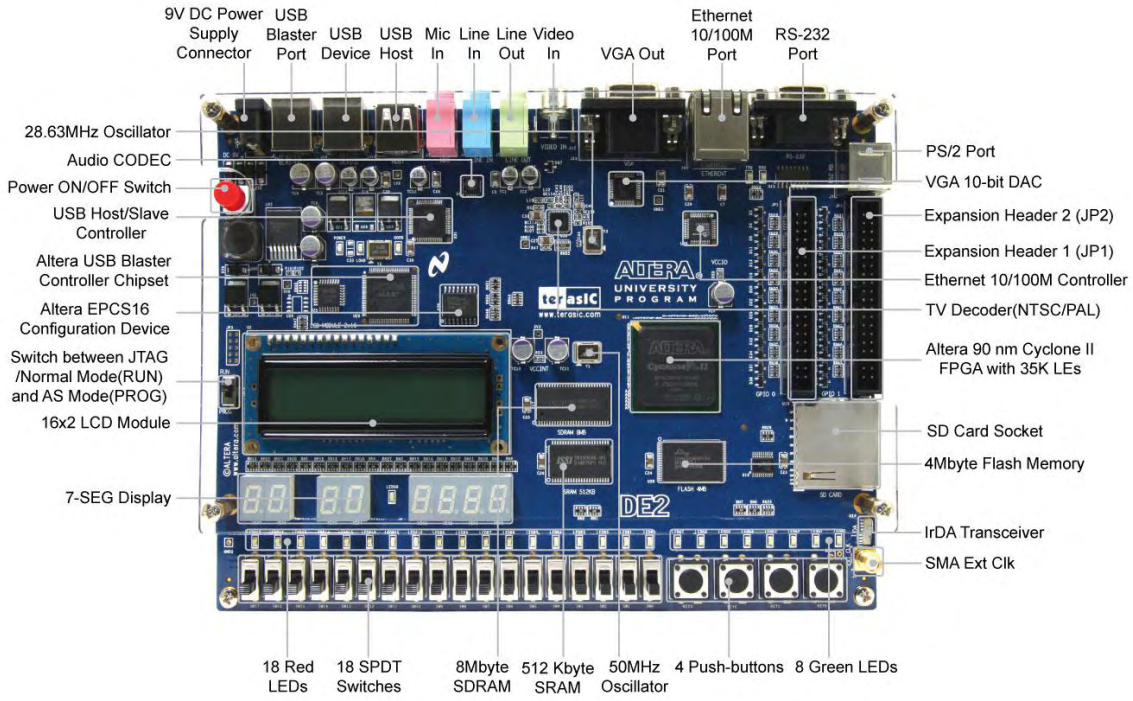
- 2 conectores externos o de expansión de 40 pines cada uno, con 72 pines de entrada/salida y 8 de alimentación o tierra.

---

[29] Ibíd, MAXFIELD, Clive. The Design Warrior's Guide to FPGAs: Devices, Tools and Flows. Elsevier, 2004. ISBN: 978-07-5067-604-5.

La tarjeta Altera DE2 cuenta con otros componentes más, conectados al dispositivo programable. Entre estos se encuentran memorias internas (512 KByte SRAM, 8 Mbyte SDRAM, 4 MByte FLASH), salida VGA, codificadores de audio y video, puerto RS-232 y PS2, controladores USB y Ethernet, etc[30].

Figura 2. Componentes de la tarjeta de desarrollo Altera DE2



Fuente: ALTERA CORPORATION. Nios II Classic Software Developer's Handbook. San Jose, CA: Altera Corporation (2015).

1.8.3. Adaptabilidad de la tarjeta para el proyecto

La tarjeta de desarrollo Altera DE2 y sus periféricos de entrada y salida, y en partícula su decodificador de video, hacen que el dispositivo inicialmente contenga los elementos fundamentales para el diseño del sistema. Se hace necesario el estudio y simulación de los algoritmos necesarios para el procesamiento del video, reconocimiento y seguimiento facial para determinar el consumo de recursos del sistema y lograr la mayor eficiencia del dispositivo y sus elementos.

---

[30] ALTERA CORPORATION. Development and Education Board User Manual. Altera Corporation (2006).

Adicionalmente se debe mencionar que la suite del programa Quartus II que acompaña a la tarjeta de desarrollo ofrece diferentes herramientas que ayudan al desarrollo del proyecto, las cuales se utilizan según el tipo de implementación (software o hardware) que se requiera.

2. DESARROLLO

2.1. DISEÑO GENERAL DEL SISTEMA

2.1.1. Requerimientos del diseño

De acuerdo a los objetivos y alcances planteados, se establecen los siguientes requisitos básicos del prototipo.

- El prototipo podrá utilizar una fuente de video con entrada NTSC a través del conector RCA.

- El procesamiento de imagen se hará en tiempo real, las operaciones se aplicarán directamente al flujo de datos.

- Los bloques de procesamiento de video permitirán un control básico para el usuario final a través de los elementos de entrada en la tarjeta de desarrollo, esto a través de un menú sencillo que permita la configuración de los parámetros.

- Los valores de configuración y conteo del sistema se mostrarán en los elementos de salida de la tarjeta, como la pantalla LCD y displays de 7 segmentos.

- Se incluirán los componentes necesarios para visualizar la imagen capturada y procesada utilizando un monitor conectado a la salida VGA de la tarjeta de desarrollo.

2.1.2. Proceso de diseño

Como inicio del proceso de diseño se realiza un diagrama de bloques funcional del sistema, cada uno de estos bloques se procede a implementar dentro la FPGA, teniendo en cuenta sus ventajas y limitaciones, aprovechando al máximo las herramientas provistas por el fabricante con el fin de realizar un prototipo funcional e independiente, es decir, que no requiera de un sistema externo para su funcionamiento.

En la figura 17 se ilustra el diagrama general para el desarrollo del proyecto. A continuación, se realiza una descripción de cada uno de sus elementos:

- Cámara para la captura de video: La cámara se encarga de capturar las señales de entrada (video) para ser procesadas por el prototipo. Su salida debe ser compatible con los dispositivos de entrada de la tarjeta de desarrollo Altera DE2, para el caso en concreto es necesario una salida de video análoga con conector adaptable a RCA. Adicionalmente se requiere que la codificación de video de la cámara sea compatible con el dispositivo decodificador de la tarjeta de desarrollo, es decir codificación de video para televisión estándar.

- Entrada de video: La tarjeta de desarrollo cuenta con una entrada de video análoga para conexión RCA así como un decodificador de video ADV 7180 para el procesamiento de video.

- Procesamiento de imagen: Consiste en el agrupamiento de filtros y módulos que preparan o mejoran los resultados de la detección para su posterior análisis y manipulación.

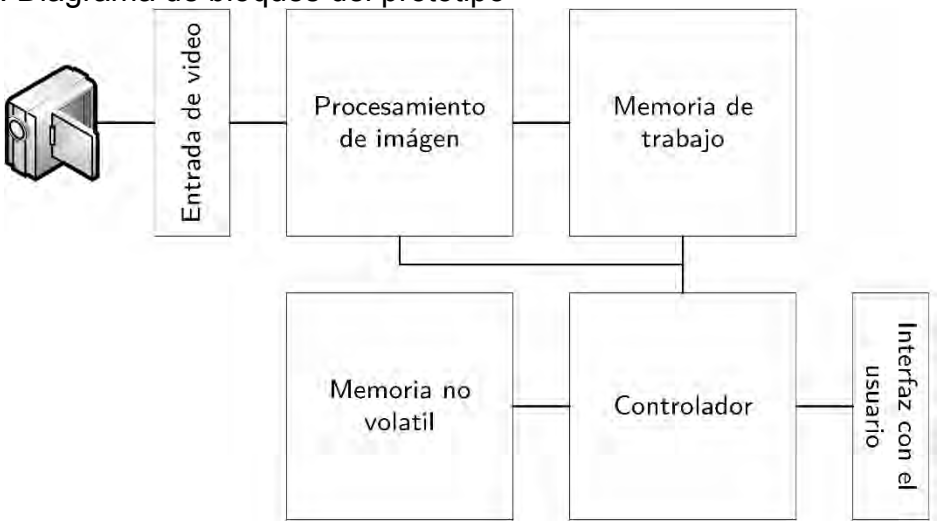
- Memoria de trabajo: Los algoritmos implementados dentro aplicación de software requieren de memoria para almacenar los datos temporales y sus resultados, para este fin se utiliza la SDRAM de la tarjeta de desarrollo.

- Controlador: El procesador Nios II, dentro de la FPGA es el encargado de gestionar la totalidad del sistema, para este fin, el procesador tiene acceso a todas las localizaciones de memoria, incluyendo los registros de los componentes personalizados para de esta forma recibir y enviar información.

- Memoria no volátil: La aplicación de software que administra la generalidad del sistema necesita estar programada en la memoria no volátil de tarjeta, en este caso la memoria Flash, con el fin de permanecer aunque el prototipo sea desconectado y mantener la configuración del programa.

- Interfaz con el usuario: A través de los dispositivos de entrada y salida de la tarjeta de desarrollo Altera DE2, se crea un menú para configuración del prototipo por parte del usuario de ciertas características que pueden variar, pues son intrínsecas del ambiente en el cual se sitúe el dispositivo. Para lo anterior se dispondrá del display LCD para visualizar el menú de configuración, de los displays de 7 segmentos para mostrar el conteo de ingreso, y los *switches* y *pushbuttons* para la navegación en el menú y configuración del prototipo.

Figura 3. Diagrama de bloques del prototipo



Fuente: Este trabajo.

2.2. ADAPTACIÓN DE COMPONENTES INCLUIDOS CON LA TARJETA DE DESARROLLO

2.2.1. Adquisición de video análogo.

Con el fin de aprovechar al máximo las características de la tarjeta de desarrollo Altera DE2, y dotar de mayor accesibilidad y economía al prototipo, se opta por utilizar el dispositivo de captura de video análogo incluido con la tarjeta.

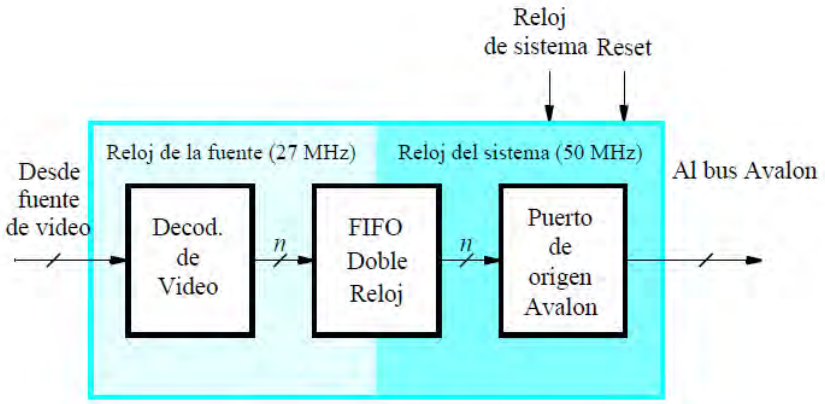
La tarjeta está equipada con un decodificador de video para televisión estándar ADV7180 fabricado por Analog Devices. Este es un decodificador de video integrado que detecta automáticamente y convierte una señal de televisión estándar en banda base (en formato NTSC, PAL, o SECAM) a formato de video de componentes (ITU-R BT.656 o ITU-R BT.601) con submuestreo de crominancia 4:2:2. Gracias al amplio uso de los estándares de video análogos como NTSC, el uso de este estándar lo hace compatible con una gran cantidad de dispositivos como reproductores de DVD, reproductores de cinta, fuentes de emisión y cámaras de seguridad[31].

Para utilizar el chip decodificador dentro de un sistema embebido Qsys, se incluye el núcleo de entrada de video (Video In), este núcleo toma la señal de salida del dispositivo ADV7180 y la sincroniza desde el dominio de reloj del mismo al dominio de reloj del sistema utilizando un FIFO asincrónico como se muestra en la Figura 4. Este FIFO es una memoria implementada dentro de la FPGA que

[31] Ibíd.

permite al puerto del bus Avalon enviar paquetes a la velocidad del sistema de 50 MHz, cuando el FIFO se encuentra vacia, el FIFO mantiene el dato anterior hasta que uno nuevo sea agregado por el decodificador de video.

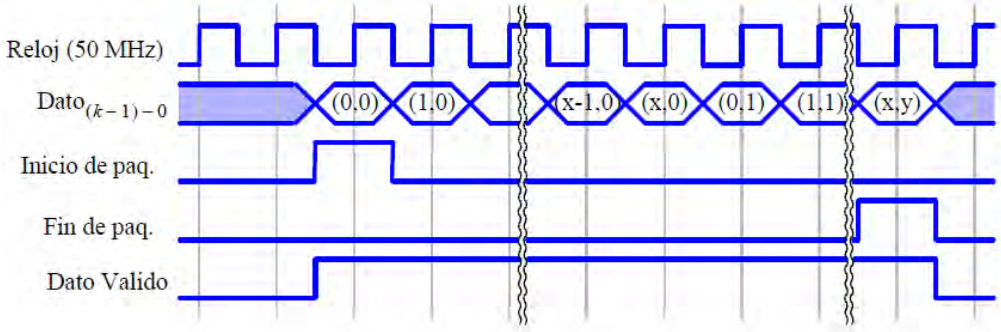
Figura 4. Diagrama de bloques del núcleo decodificador de video



Fuente: ALTERA CORPORATION. Video IP Cores for Altera DE-Series Boards. San Jose, CA: Altera Corporation (2013), p. 1-47.

La fase salida del núcleo convierte las señales de video al formato *Altera University Program* a través de una interfaz Avalon Streaming (Avalon-ST). En este formato, cada paquete en el flujo representa un cuadro de video. Los cuadros se transmiten un pixel a la vez, el primer pixel ubicado en la parte superior izquierda de la imagen es señalizado por el bit de inicio de paquete mientras que el último pixel es señalizado por el bit de fin de paquete de la interfaz Avalon-ST.

Figura 5. Diagrama de tiempo para un paquete de video en formato *Altera University Program* en la interfaz Avalon-ST

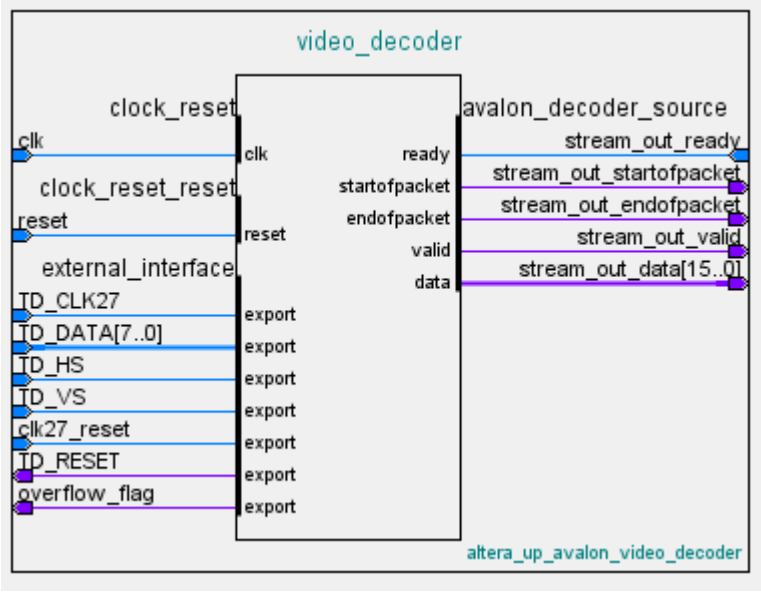


Fuente: ALTERA CORPORATION. Video IP Cores for Altera DE-Series Boards. San Jose, CA: Altera Corporation (2013), p. 1-47.

El formato de cada pixel depende del espacio de color del cuadro de video, algunos núcleos no requieren información sobre el espacio de color de los pixeles, para los núcleos que requieren conocerlo es importante saber el número de bits por color, el número de planos de color de los paquetes entrantes. En el caso del

núcleo de entrada de video, la salida es de 720 columnas por 244 filas con 16-bits por pixel en el espacio de color 4:2:2 YCbCr en modo interlineado[32].

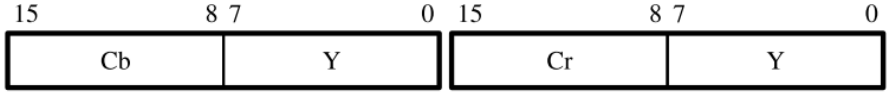
Figura 6. Entradas y salidas del módulo decodificador de video.



Fuente: ALTERA CORPORATION. Video IP Cores for Altera DE-Series Boards. San Jose, CA: Altera Corporation (2013), p. 1-47.

El espacio 4:2:2 YCbCr alterna los componentes de crominancia de cada pixel consecutivo, la Figura 7 muestra la representación de este modo en el formato *Altera University Program* y se define como 8 bits por color y 2 planos de color para un total de 16 bits por pixel. Esta trama de datos es enviada a través del bus Avalon (puerto `stream_out_data`) para ser procesada por el siguiente bloque.

Figura 7. Representación del pixel con espacio de color YCbCr 4:2:2



Fuente: ALTERA CORPORATION. Video IP Cores for Altera DE-Series Boards. San Jose, CA: Altera Corporation (2013), p. 1-47.

2.2.2. Conversión de espacio de color.

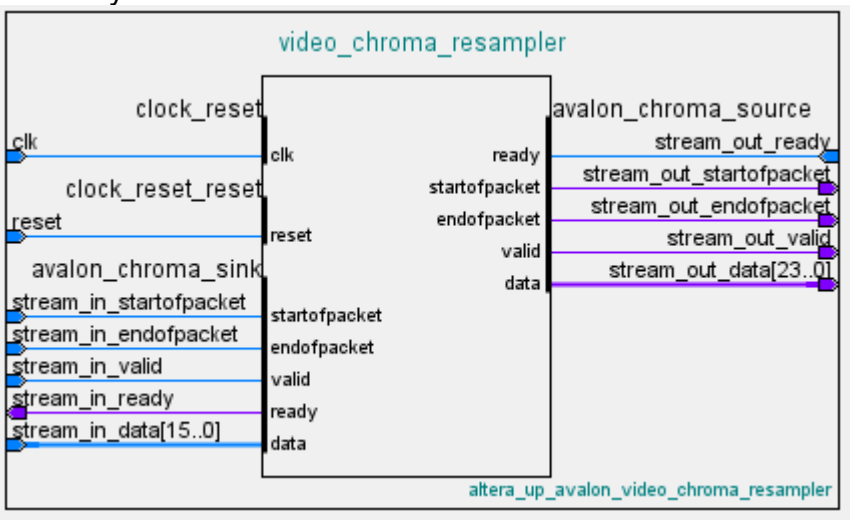
El espacio de color YCbCr 4:2:2 esta optimizado para el ahorro de imagen, sin embargo, alternar el color transmitido en cada paquete hace que el procesamiento

---

[32] ALTERA CORPORATION. Video IP Cores for Altera DE-Series Boards. San Jose, CA: Altera Corporation (2013).

sea más complejo, por esto se transforma la imagen al espacio de color YCbCr 4:4:4 que mantiene la resolución completa en los componentes de luminancia y crominancia, simplificando el procesamiento y permitiendo el ahorro de recursos de hardware.

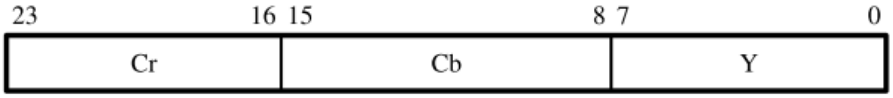
Figura 8. Entradas y salidas del módulo de remuestreo de croma.



Fuente: ALTERA CORPORATION. Video IP Cores for Altera DE-Series Boards. San Jose, CA: Altera Corporation (2013), p. 1-47.

El módulo de remuestreo de croma recibe un flujo organizado como lo muestra la Figura 7, este almacena el componente Cb o Cr en un registro de forma alternada y lo escribe en la salida para formar un flujo con el orden de acuerdo a la Figura 9. La salida de este proceso es una imagen de 3 planos de color, con 8 bits por plano, para un total de 24 bits como se muestra en la Figura 9.

Figura 9. Representación del pixel con espacio de color YCbCr 4:4:4



Fuente: ALTERA CORPORATION. Video IP Cores for Altera DE-Series Boards. San Jose, CA: Altera Corporation (2013), p. 1-47.

2.2.3. Visualización a través de VGA.

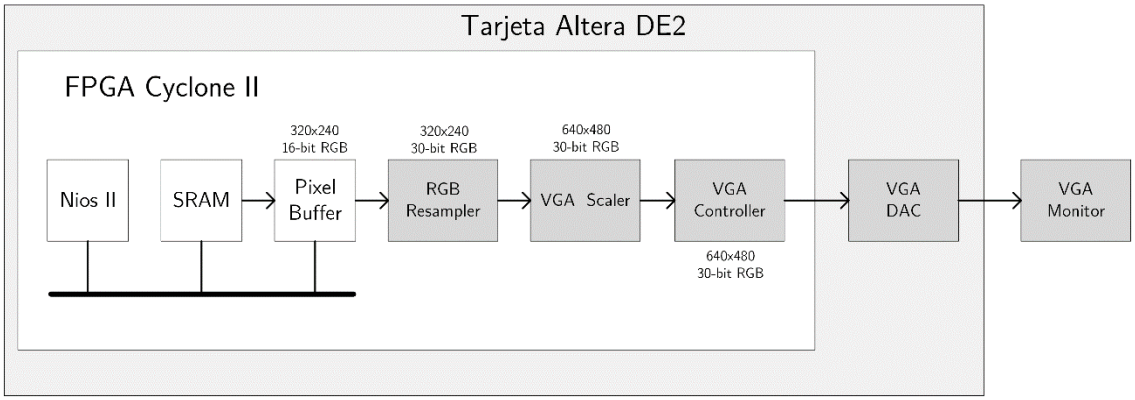
Con el fin de depurar los algoritmos realizados, se aprovechan las características de salida de video la tarjeta de desarrollo DE2, con el fin de observar en un monitor VGA, los resultados de las operaciones de imagen aplicadas.

VGA requiere de un manejo de tiempo preciso, el cual se facilita por las herramientas provistas por el fabricante. El estándar requiere de señales RGB en forma análoga y cada pixel debe estar sincronizado con un ciclo de reloj de 25,175MHz[33].

Ya que las señales de entrada y salida se procesan a frecuencias diferentes se requiere el uso de un búfer de pixeles. Este almacena y envía los valores de color de los pixeles a mostrar y a través de un controlador DMA, los envía utilizando la interfaz Avalon ST[34].

La señal de salida del búfer debe ajustarse a las características requeridas por el estándar VGA a resolución de 640 x 480 (es decir, es necesario doblar la resolución de la imagen procesada usando interpolación proximal) como se observa en la Figura 10 y dado que la información de color se transmite de forma análoga, la profundidad de color es de 10 bits, tal como lo requiere el DAC de video ADV7123 incluido en la tarjeta de desarrollo[35].

Figura 10. Esquema del sistema de salida VGA



Fuente: Este trabajo

La implementación de estas características se realiza utilizando los bloques IP de *University Program*, ajustados a la resolución y características de imagen procesadas, con el fin de entregar al controlador VGA una resolución adecuada pero que mantenga la imagen sin modificaciones adicionales.

---

[33] HAMBLEN, James O., et al. Rapid Prototyping of Digital Systems. Springer, 2008. ISBN: 978-03-8772-670-0.
[34] ALTERA CORPORATION, Development and Education Board User Manual. Op. cit.
[35] HAMBLEN, et al. Op. cit, ALTERA CORPORATION, Development and Education Board User Manual. Op. cit.

La interacción entre el procesador y el hardware implementado se realiza utilizando la técnica de doble buffer, es decir, un espacio de memoria dedicado al almacenamiento temporal de imágenes que permite la edición de las imágenes sin afectar la sincronización con la salida de video.

2.2.4. Buffer de pixeles.

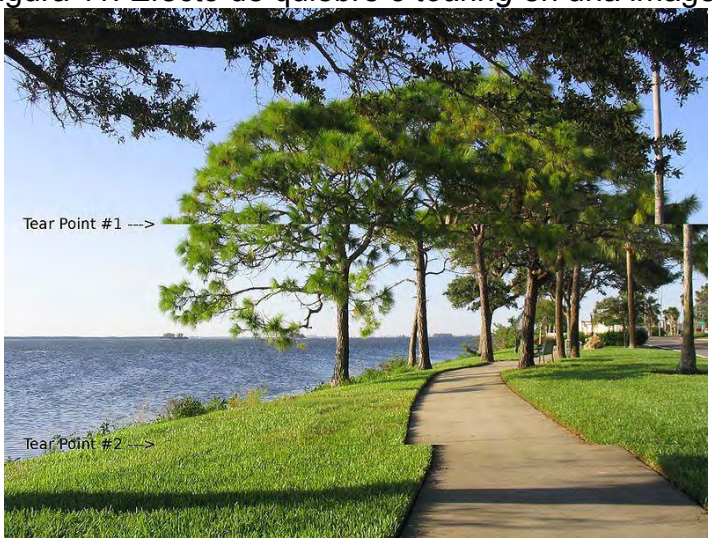
El búfer de pixeles es un módulo de memoria dedicada, en el caso de la tarjeta Altera DE2 se utiliza la memoria SRAM, esta almacena los cuadros provenientes del módulo DMA de entrada, con espacio de color Y, y los almacena utilizando un direccionamiento consecutivo empezando por el pixel de la esquina superior izquierda (0,0), ocupando así un espacio de 614.400 bits o 76,8 KBytes de memoria. El uso de la SRAM sobre la SDRAM obedece a la velocidad alcanzada por esta, lo que lo hace apta para aplicaciones en tiempo real.

2.2.5. DMA del buffer de pixeles.

El módulo DMA o acceso directo a memoria es un dispositivo que usa la interfaz Avalon MM para leer los pixeles almacenados en el búfer y los transmite usando el bus Avalon ST exponiendo una interfaz tipo *source* que envía los datos a otros bloques Avalon ST. Este módulo también expone una serie de registros al procesador a través de su interfaz Avalon MM esto le permite modificar la imagen a mostrar sin cambiar el contenido del pixel a través del concepto del búfer doble (*double-buffering*) en el cual se usan dos búferes, llamados búfer delantero y trasero.

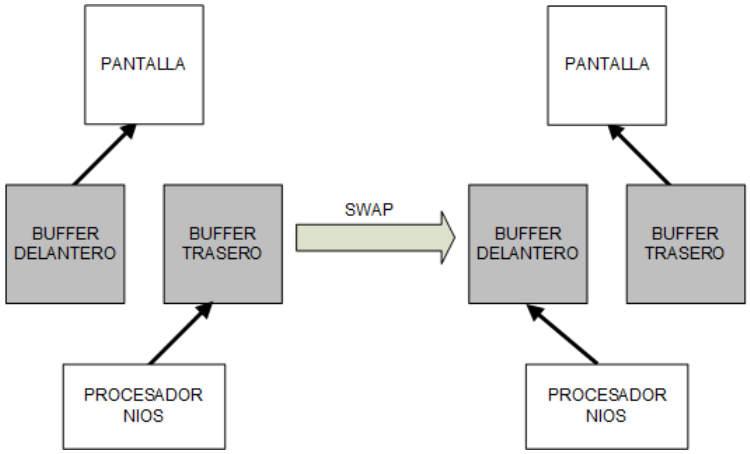
Estos búferes aseguran que la pantalla siempre tiene un cuadro completo para dibujar, y permite al procesador modificar los cuadros sin causar desincronización de la imagen lo que genera un efecto de “quiebre” en la imagen, conocido técnicamente como *tearing* y mostrado en la . El esquema de funcionamiento para el método del búfer doble se muestra en la Figura 12.

Figura 11. Efecto de quiebre o *tearing* en una imagen



Fuente: Wikimedia Commons. A typical video tearing artifact (simulated image)

Figura 12. Esquema de funcionamiento del método de búfer doble (*double-buffering*)



Fuente: Este trabajo

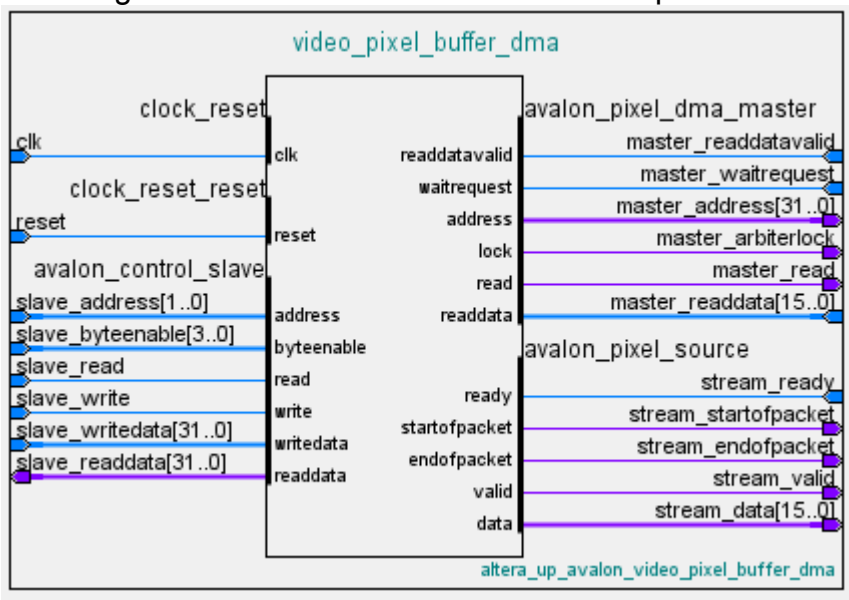
El proceso de intercambio o *swap* entre los contenidos de los búferes delantero y trasero se realiza escribiendo cualquier valor al registro búfer. El módulo interpreta esta operación como inicio de la transferencia, esta no es instantánea, en cambio, ocurre al finalizar el dibujado del cuadro actual, asegurando así la transferencia total del mismo. El estado del proceso se vigila leyendo el bit S del registro estado. Cuando la transferencia está en curso el bit toma el valor de 1, y cuando finaliza cambia a 0.

El funcionamiento del controlador es el siguiente: mientras la imagen contenida en el búfer de pixeles es mostrada, una nueva imagen se dibuja dentro búfer trasero. Cuando esta imagen esta lista para ser mostrada se realiza un intercambio. Luego

el búfer de pixeles al que ahora apunta el registro del búfer trasero, que ya fue mostrado, se borra y se dibuja la nueva imagen. Este intercambio se realiza a una velocidad mucho mayor que la velocidad de cuadros del video (29,997 cps) por lo que el procesador no tiene problema en acceder y modificar la imagen.

Este módulo permite al procesador acceder a la imagen que se muestra como si se tratara de una memoria, y el concepto del doble búfer impide que una operación del procesador retrase la transmisión de la imagen dentro del sistema. Dentro del prototipo, se usa para mostrar el umbral de la región de detección, simbolizado con una línea de color rojo, la cual se actualiza de manera dinámica de acuerdo al valor asignado por el usuario como se muestra en la Figura 14.

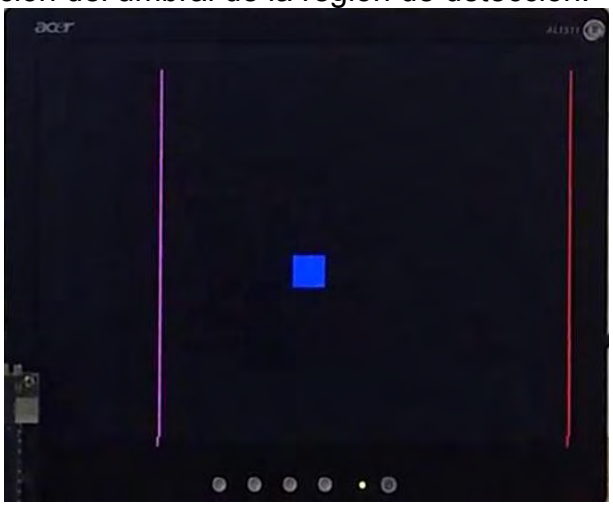
Figura 13. Módulo de DMA del búfer de pixeles.



Fuente: Este trabajo.

La conexión de este módulo, mostrada en la Figura 13, se realiza a través del bus Avalon MM, el esclavo de control `avalon_control_slave` es controlado por el procesador Nios II y permite acceder a los espacios de memoria SRAM, la cual es controlada por el maestro DMA `avalon_pixel_dma_master`. El flujo de imagen que sale hacia el control VGA se hace a través del bus Avalon ST, controlado por `avalon_pixel_source`.

Figura 14. Visualización del umbral de la región de detección.



Fuente: Este trabajo

2.3. DISEÑO DE COMPONENTES PERSONALIZADOS

2.3.1. Segmentación de imagen.

La segmentación basada en color es un método muy útil y sencillo para la implementación facial, en particular, varias investigaciones muestran la idoneidad de la segmentación basada en el espacio de color YCbCr[36] y aunque el espacio de color CIELab obtiene los mejores resultados[37], este es un espacio colorimétrico, cuyas transformaciones no son adecuadas para una implementación en un sistema digital[38].

Para la segmentación se tienen en cuenta únicamente los canales de croma Cb y Cr, con el fin de minimizar la influencia del brillo de la imagen y hacer que el color

---

[36] AL-TAIRI, Zaher Hamid, et al. Skin segmentation using YUV and RGB color spaces. En: Journal of Information Processing Systems. jun, 2014. vol. 10

[37] SHIN, Min C., et al. Does colorspace transformation make any difference on skin detection? En: Proceedings of IEEE Workshop on Applications of Computer Vision (2002-Janua). IEEE, 2002, ALBIOL, Alberto, et al. Optimum color spaces for skin detection. En: Journal of Chemical Information and Modeling. 1989. vol. 53

[38] THAKUR, Sayantan, et al. Face detection using skin tone segmentation. En: 2011 World Congress on Information and Communication Technologies. IEEE, 2011, KAUR, Amanpreet y B. V. KRANTHI. Comparison between YCbCr Color Space and CIELab Color Space for Skin Color Segmentation. En: International Journal of Applied Information Systems. 2012. vol. 3

sea el criterio más importante para la segmentación[39]. Teniendo en cuenta los resultados experimentales en la investigación de Berbar et al[40], se utiliza la fórmula descrita inicialmente por Hu et al[41]. Cualquier pixel que satisfaga con las inecuaciones siguientes se considera un pixel de piel:

$$137 < \square\square < 177$$
$$77 < \square\square < 127$$
$$190 < \square\square + 0{,}6\square\square < 215$$

El uso de los componentes de crominancia evita que la información de iluminación influya sobre la segmentación. La intensidad de los pixeles no aporta información útil para la detección basada en color, por tanto, el componente de luminancia Y no se usa para este bloque[42].

La implementación de las dos primeras inecuaciones puede realizarse en VHDL utilizando comparadores, sin embargo, la tercera inecuación al involucrar una multiplicación por un número decimal requiere de un análisis más detallado con el fin de evitar el uso de aritmética de punto flotante. Esto se realiza multiplicando la inecuación por 256 obteniéndose así:

$$48640 < 256\square\square + 153\square\square < 55040$$

El multiplicar por 256 o 2^8 se realiza con el fin de escalar la operación para que sea realizada por el multiplicador de enteros con entradas de 8 bits incluido en la tarjeta de desarrollo, de esta manera la operación se realiza en un solo ciclo de reloj sin utilizar más que los multiplicadores dedicados, permitiendo un uso mínimo de recursos y evitando el aumento de la latencia en las operaciones del bloque.

Tabla 1. Características de la implementación de multiplicadores LPM

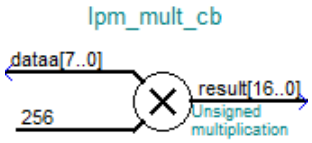
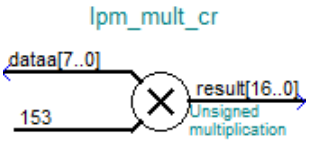
	Multiplicador Cb	**Multiplicador Cr**

[39] BIN ABDUL RAHMAN, Nusirwan Anwar, et al. RGB-H-CbCr Skin Colour Model for Human Face Detection. En: Proceedings of The MMU International Symposium on Information & Communications Technologies (M2USIC 2006). 2006

[40] BERBAR, et al. Op. cit.

[41] HU, M., et al. A Fast and Efficient Chin Detection Method for 2-D Scalable Face Model Design. En: International Conference on Visual Information Engineering, 2003. VIE 2003. 2003

[42] BASILIO, Jorge Alberto Marcial, et al. Explicit Image Detection using YCbCr Space Color Model as Skin Detection. En: American Conference on Applied Mathematics (AMERICAN-MATH '11) - 5th WSEAS International Conference on Computer Engineering and Applications (CEA '11) (Puerto Morelos, Mexico). 2011

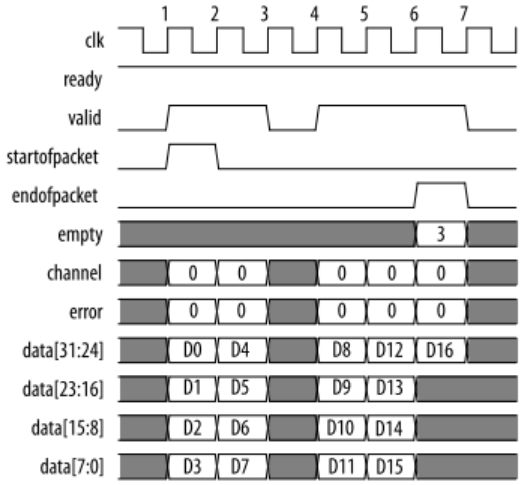
	Multiplicador Cb	Multiplicador Cr
Diagrama esquemático		
Tipo de multiplicación	Sin signo (*Unsigned*)	
Ancho de entrada	8 bits	
Ancho de salida	17 bits	

Fuente: Este trabajo.

Una vez obtenida la función escalada, es posible realizar las comparaciones requeridas como se muestra en la Figura 16. El resultado final es una imagen binaria, la cual tiene exclusivamente el canal Y, para los pixeles que corresponden a un pixel de piel según el segmentador, se les asigna el valor máximo de 255 mientras que a los demás se les asigna 0.

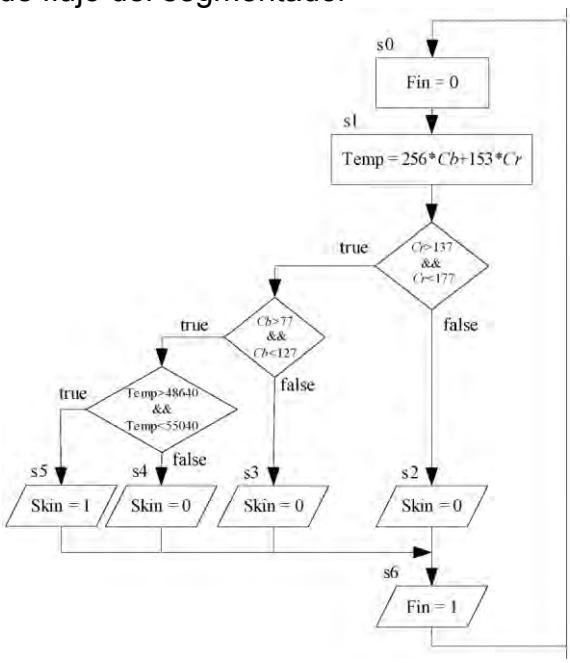
Los datos de salida deben transmitirse de acuerdo a las especificaciones y tiempos del bus Avalon ST como se muestra en la Figura 15, este implementa en una máquina de estado que toma las señales de entrada y las almacena en un registro con el fin de ser procesadas. Para el caso de los registros de salida, el resultado del componente personalizado se conecta a estos registros para que la máquina de estado los transmita en formato compatible con las especificaciones Avalon ST.

Figura 15. Diagrama de tiempo para el bus Avalon ST



Fuente: ALTERA CORPORATION. Avalon Interface Specifications. San Jose, CA: Altera Corporation (2015).

Figura 16. Diagrama de flujo del segmentador



Fuente: CHEN, Ching Yi, CHEN, Ching Han; HO, Hsiao Ping. A real-time skin color based human face tracking system using an evolutionary search strategy. En: Journal of Applied Science and Engineering. (2013), p. 249-259.

2.3.2. Filtrado de imagen.

El uso de núcleos de filtros espaciales para aplicaciones de filtrado de imagen ha sido la metodología estándar para los diseñadores de hardware. Por tanto, este método ha sido bastante estudiado y presenta variaciones útiles para un gran número de aplicaciones.

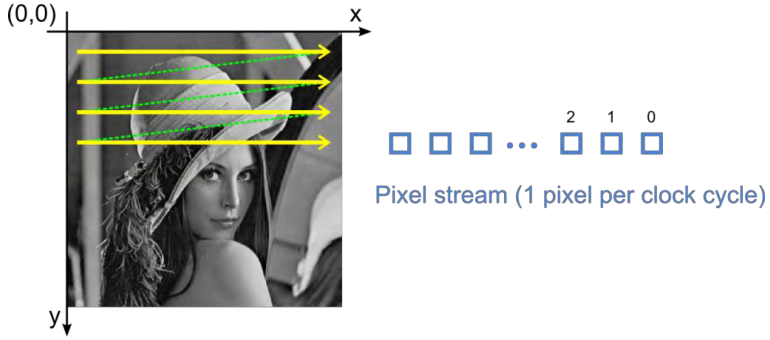
Uno de los componentes esenciales para la implementación de un filtro es el operador de ventana[43]. Los operadores de ventana usan los pixeles adyacentes para calcular la salida. Por ejemplo, implementar un filtro medio, un operador de ventana utiliza los pixeles vecinos a un pixel central para encontrar el promedio[44].

En la programación de software, las imágenes son matrices bidimensionales que residen en la memoria, los algoritmos tienen a su disposición la imagen completa, por lo que un filtro se implementaría a través de un par de ciclos "`for`" que se iteran sobre cada pixel. Sin embargo, este paradigma no es útil para la

[43] CHUCKS, U. VHDL Image Processing. 2011, NELSON, Anthony Edward. Implementation of Image Processing Algorithms on FPGA Hardware. Nashville: (2000).
[44] NELSON. Op. cit.

programación de hardware, puesto que las imágenes son simples flujos continuos de pixeles en orden de *raster* (de la parte superior izquierda a la inferior derecha), cada pixel ocupa 8 bits para una imagen en escala de grises como como se muestra en la Figura 17, y dado que las FPGA tienen componentes de memoria limitados, se almacenan líneas en vez de cuadros completos[45].

Figura 17. Flujo de imágenes en orden de escaneo *raster*

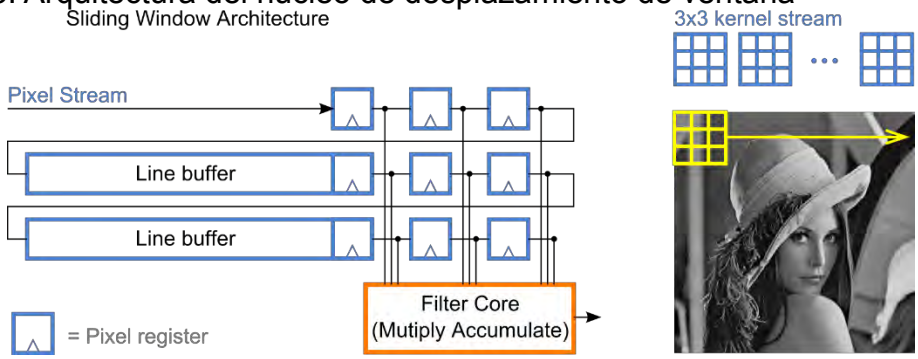


Fuente: GHANNOUM, Anthony. Image Filtering in FPGAs | Teledyne DALSA Imaging Blog. (2012).

Los búferes de línea actúan como un conducto que retrasan los flujos entrantes por un factor igual al ancho de la imagen, por lo que imágenes más grandes requieren mayor cantidad de mem oria. Por ejemplo, esto se traduce a desplazar una ventana de un tamaño dado en orden *raster* sobre la imagen. Esta ventana de 3×3 se conecta al núcleo de filtro, el cual se encarga de realizar las operaciones como se muestra en la Figura 18[46].

[45] AVIZIENIS, Rimas. Building your First Image Processing ASIC | CS250 Laboratory 2. Autor (2012), GHANNOUM, Anthony. Image Filtering in FPGAs | Teledyne DALSA Imaging Blog. (may, 2012).
[46] GHANNOUM. Op. cit, BENDA, Lukas. Hardware acceleration for image processing. (2008).

Figura 18. Arquitectura del núcleo de desplazamiento de ventana

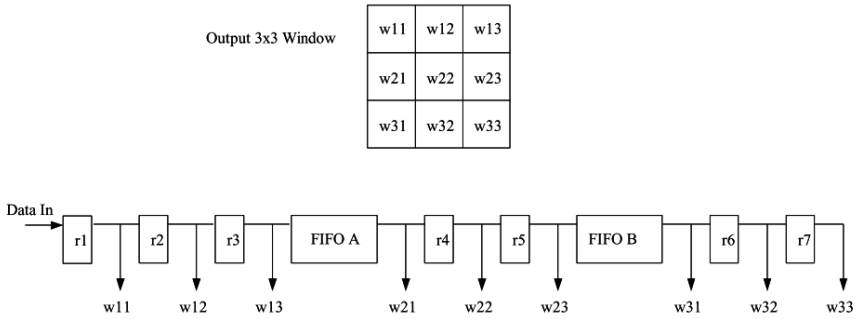


Fuente: GHANNOUM, Anthony. Image Filtering in FPGAs | Teledyne DALSA Imaging Blog. (2012).

Para el prototipo se opta por un operador de ventana de 5 × 5 para el filtro gaussiano, y 7 × 7 para los demás filtros, estos tamaños ofrecen la mejor solución para la resolución de imagen de trabajo (320 x 240), el menor tamaño para el filtro gaussiano es debido a la complejidad de las operaciones involucradas, mientras que los filtros que operan sobre la imagen binaria requieren de muchos menos recursos en la FPGA[47].

En el caso de los pixeles que se encuentran en los bordes y esquinas de la imagen, los valores que quedan por fuera de la ventana son los valores inicializados en el buffer de línea; estos se inicializan con un valor de 0, como si la imagen estuviera rodeada de pixeles con este valor, esto para evitar afectar la región de interés de la imagen.

Figura 19. Implementación de un operador de ventana de 3 × 3



Fuente: NELSON, Anthony Edward. Implementation of Image Processing Algorithms on FPGA Hardware. Nashville: (2000), p. 79.

[47] BENDA. Op. cit.

2.3.3. Filtro de mediana.

Utilizando la implementación del método de desplazamiento de ventana es posible implementar un filtro de mediana con el propósito de ayudar a mejorar el resultado de la segmentación, pues dicho filtro remueve tanto el ruido como el detalle en particular el ruido conocido como "sal y pimienta" como se muestra en la Figura 20, pues para nuestro objetivo no es necesario contar con una imagen detallada[48].

Figura 20. Ejemplo de ruido "sal y pimienta" y la imagen después de filtrarla con un filtro de mediana.

Fuente: Este trabajo

La implementación del filtro de mediana en la FPGA utiliza el método de operador de ventana descrito anteriormente, para poder operar con los valores de los cuadros que rodean al pixel actuar, se usan búferes FIFO para almacenar las líneas previas y siguientes del cuadro, estos búferes se implementan en registros de corrimiento de 8 bits de ancho por 320 de largo, y se conectan como se muestra en la Figura 21

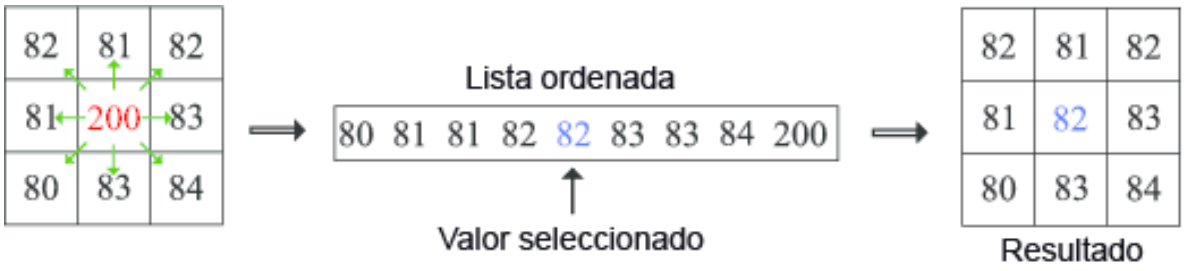
En el caso de los pixeles que se encuentran en los bordes y esquinas de la imagen, los valores que quedan por fuera de la ventana son los valores inicializados en el buffer de línea; estos se inicializan con un valor de 0, como si la imagen estuviera rodeada de pixeles con este valor, esto para evitar afectar la región de interés de la imagen

Los valores dentro del búfer se comparan entre sí para establecer una lista ordenada de valores, el valor central de esta lista corresponde al valor de la mediana de los pixeles dentro de la ventana como se muestra en la Figura 21.

---

[48] SERFA JUAN, Ronnie O., et al. FPGA Implementation for Real Time Sobel Edge Detector Block Using 3-Line Buffers. En: International Journal of Industrial Electronics and Electrical Engineering. dec, 2015. vol. 3

Figura 21. Esquema de funcionamiento del filtro de mediana para una ventana de 3 x 3.

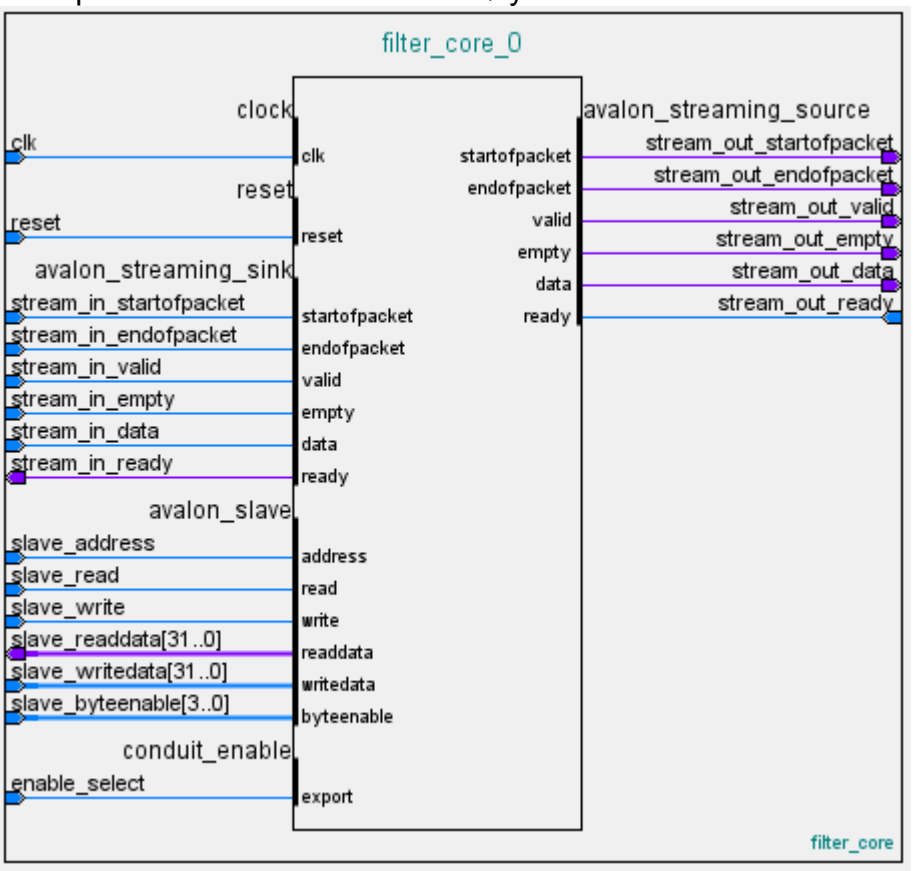


Fuente: Este trabajo.

Para el caso de imágenes binarias, teniendo en cuenta que los valores se limitan a 0 o 1, un filtro de mediana es equivalente a establecer que los valores dentro de la ventana, suman menos de la mitad truncada más 1 de los espacios disponibles, para el caso del prototipo, que usa un filtro de mediana de 7 x 7, es equivalente a que la suma de los valores sea menor a 25.

Durante las pruebas se observó que un tamaño de ventana más grande eliminaría objetos de mayor tamaño, para la imagen del prototipo de resolución de 320 x 240 usar un filtro de 7 x 7, elimino objetos que no influían en la detección facial, como ruido de iluminación generado por el segmentador y objetos detectados que no corresponden a piel, pero son muy pequeños para ser considerados como tal.

Figura 22. Bloque del filtro de mediana en Qsys



Fuente: Este trabajo.

El componente personalizado que implementa estas funcionalidades se llama `filter_core`, este componente puede ser controlado por un interruptor encendido o apagado a través de la entrada `enable_select`). Sus entradas y salidas funcionan de manera que sean compatibles con el bus Avalon ST.

2.3.4. Filtro gaussiano.

Antes de segmentar la imagen, se realiza un procesamiento previo de la misma a través de un filtro gaussiano, el uso de este filtro elimina visiblemente el ruido causado por el sensor de la cámara CMOS, a costa del detalle de la imagen. Ya que los detalles de la imagen no son de la misma relevancia que el color se encuentra conveniente el uso de este filtro.

El filtro gaussiano implementado se basa en el siguiente núcleo de 5 × 5 con $\sigma = 1{,}4$, con el fin de evitar utilizar operaciones de punto flotante, se multiplica el

núcleo por 159 y se redondea, de esta forma la mayoría de las multiplicaciones pueden implementarse con simples corrimientos[49]:

$$□_□ = \frac{1}{128}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Utilizando la operación de ventana es posible multiplicar cada uno de los valores obtenidos por el núcleo mostrado, para así realizar la operación de convolución entre los valores de la imagen y los valores del núcleo.

Para evitar el uso de multiplicadores dedicados y simplificar el diseño y la implementación. En la Tabla 2, se muestran las ecuaciones que detallan la equivalencia entre la multiplicación por los factores del núcleo y su implementación como operaciones de corrimiento y suma, de esta forma se evita la necesidad de utilizar multiplicadores dedicados, ahorrando recursos de la FPGA.

Tabla 2. Implementación de las operaciones dentro del lenguaje VHDL

Factor	**Implementación**
Multiplicación por 2	$□ * 2 = (□ \ll 1)$
Multiplicación por 4	$□ * 4 = (□ \ll 2)$
Multiplicación por 5	$□ * 5 = □(2^2 + 1)$ $□ * 5 = □ * 2^2 + □$ $□ * 5 = (□ \ll 2) + □$
Multiplicación por 9	$□ * 9 = □(2^3 + 1)$ $□ * 9 = □ * 2^3 + □$ $□ * 9 = (□ \ll 3) + □$
Multiplicación por 12	$□ * 12 = □(2^3 + 2^2)$ $□ * 12 = □ * 2^3 + □ * 2^2$ $□ * 12 = (□ \ll 3) + (□ \ll 2)$
Multiplicación por 15	$□ * 15 = □(2^4 - 1)$ $□ * 15 = □ * 2^4 - □$ $□ * 15 = (□ \ll 4) - □$

[49] RASHID, Syed Sameer, et al. VHDL Based Canny Edge Detection Algorithm. En: International Journal of Current Engineering and Technology. 2014. vol. 4, no. 2

Factor	Implementación
División por 159	$\frac{□}{128} = (□ \gg 5)$
Operadores: * - Multiplicación << - Corrimiento a la derecha >> - Corrimiento a la izquierda / - División	

Fuente: Este trabajo

Al final se suman los resultados de las multiplicaciones obteniéndose el cómo salida el valor del pixel filtrado. Para evitar desbordamiento de los resultados, las entradas son valores de intensidad de 8 bits, mientras que el resultado se le asignan 16 bits.

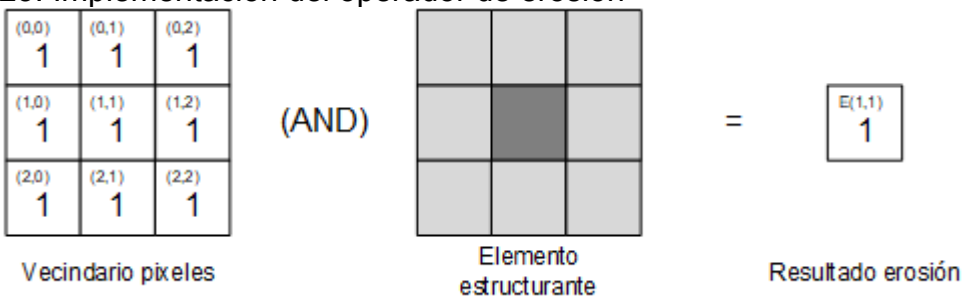
El uso del filtro mejora levemente los resultados, en particular en áreas con variación de iluminación, el cual es causado por el sensor CMOS, así mismo, suaviza los bordes del segmentador.

2.3.5. Filtros morfológicos.

Las operaciones morfológicas de erosión y dilatación están relacionadas con la convolución, sin embargo, estas utilizan operaciones lógicas en vez de aritméticas. Las operaciones morfológicas binarias combinan un vecindario de pixeles con una máscara para lograr el resultado. El pixel de salida se establece en 0 o 1 basado en el resultado de la operación lógica AND entre todos los elementos de la ventana.

Para la implementación se usa una máscara o elemento estructurante cuadrado como se muestra en la Figura 23.

Figura 23. Implementación del operador de erosión



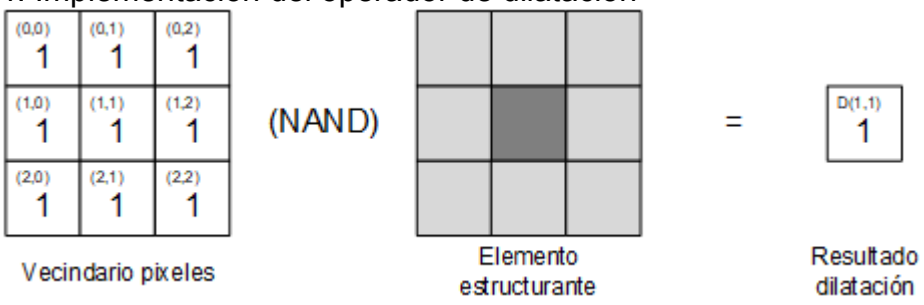
Fuente: Este trabajo.

Esto quiere decir que para que la salida sea 1, todos los pixeles dentro del vecindario deben ser 1, de otra manera el pixel será 0. De hecho, la presencia de un valor 0 dentro del vecindario hará que la salida sea 0. La erosión se puede usar

para eliminar los pixeles de ruido no deseados de un área mayoritariamente negra[50].

La dilatación es el opuesto de la erosión, este proceso hace que las áreas blancas crezcan, o se dilaten; para su implementación se invierta la lógica, en vez de usar compuertas AND se usa una compuerta NAND[51] y su elemento estructurante se muestra en la Figura 24.

Figura 24. Implementación del operador de dilatación



Fuente: Este trabajo.

La combinación de ambas operaciones produce una operación de apertura, la cual remueve las regiones que no pueden contener al elemento estructurante, suaviza los contornos, elimina conexiones delgadas y remueve protrusiones delgadas. El objetivo de este proceso es obtener una imagen representativa de las regiones de piel sin ruido y elementos ajenos[52].

Para implementar las operaciones de apertura y cerramiento se implementan las operaciones fundamentales de erosión y dilatación dentro de un componente personalizado llamado `morphological_core`, este componente puede ser controlado por dos interruptores, los cuales controlan el estado (encendido o apagado a través de la entrada `enable_selec`) y la operación (erosión o dilatación a través de la entrada `mode_select`).

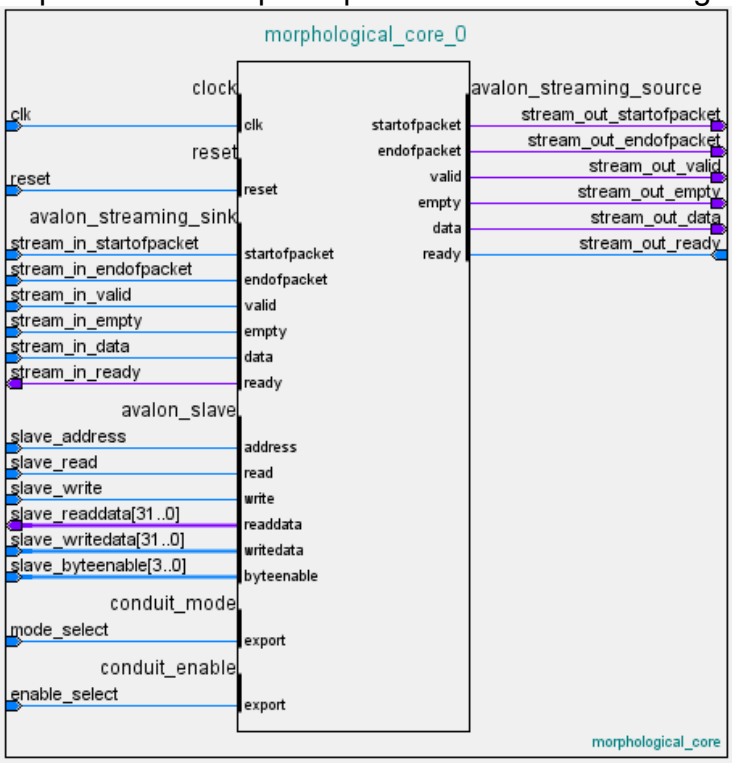
---

[50] ARUNMOZHI, R. y G. MOHAN. Implementation of Digital Image Morphological Algorithm on FPGA using Hardware Description Languages. En: International Journal of Computer Applications. 2012. vol. 57, KIM, Inseong, et al. Face Detection. En: Computer and Information Science. 2010. vol. 3

[51] ARUNMOZHI y MOHAN. Op. cit.

[52] Ibíd.

Figura 25. Núcleo personalizado que implementa el filtro morfológico.



Fuente: Este trabajo.

En el prototipo final estos controles se conectan a los interruptores de la tarjeta de desarrollo, permitiendo observar en tiempo real los efectos del filtro sobre la imagen segmentada.

2.3.6. Filtro temporal

El proceso de segmentación genera pequeños pixeles cuyo valor cambia rápidamente entre cuadros sucesivos, estos corresponden a las pequeñas variaciones de color causadas por el sensor CMOS. Para eliminar este ruido, se realiza un filtro temporal que sea capaz de promediar los valores entre cuadros sucesivos, y mostrar un valor de 1 si este promedio supera un valor de umbral de la siguiente forma:

$$□□□_{□□□} = \frac{1}{2}□□□_{□□} + \frac{1}{2}□□□_{□□□□}$$

En donde $□□□_{□□}$ corresponde al promedio del filtro temporal obtenido en el cuadro presente y $□□□_{□□□□}$ al obtenido al cuadro anterior, el valor de salida del filtro corresponde al promedio cumulativo de los dos últimos cuadros.

Para simplificar la implementación de las operaciones, la ecuación a implementar con operaciones de corrimiento corresponde a:

$$□□□_{□□□} = (□□□_{□□} \gg 1) + (□□□_{□□□□} \gg 1)$$

Una vez obtenido este promedio, el valor de salida se establece en 1 si el promedio es mayor a 0.5, para asegurar que los pixeles que tengan el valor de 1 por más de 2 cuadros, mantengan este valor, en caso contrario se establece la salida en 0.

La sintetización de este componente implica una memoria de gran tamaño y con capacidad de ser leída mientras se escribe un valor, ya que se requiere que el valor del promedio del cuadro anterior se guarde para cada pixel, por esto fue necesario limitar la representación del valor del promedio a 4 bits.

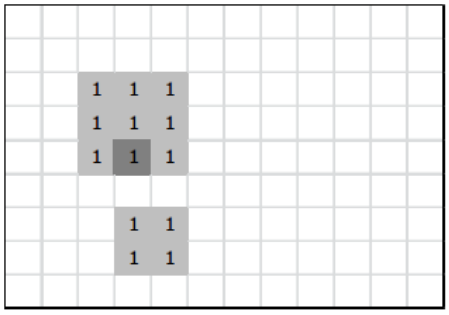
2.3.7. Cálculo del centroide.

El módulo de cálculo del centroide incluye los elementos tanto de análisis como de visualización, en primer lugar, el componente obtiene un promedio de la posición tanto en el eje x como y realizando una suma acumulada de las coordenadas de los pixeles con valor 1.

Este valor acumulado se divide entre el número de pixeles con valor 1 utilizando un bloque de división dedicado que se incluye dentro de la FPGA. De esta manera se obtiene la ubicación central de los elementos detectados.

La Figura 26 muestra que el módulo calcula el promedio de todas las regiones, razón por la cual el proceso ignora una región de 20 pixeles de los bordes de la imagen con el fin de evitar que elementos fuera del área de interés afecten el cálculo del promedio.

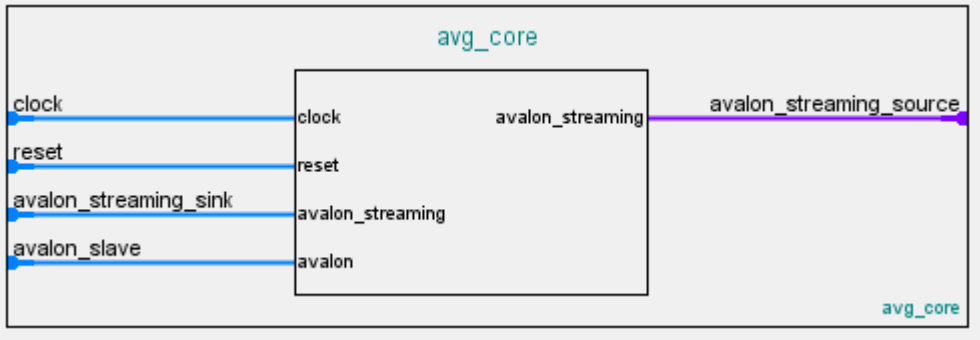
Figura 26. Cálculo del centroide de la imagen



Fuente: Este trabajo

Este módulo también se encarga de dibujar en la pantalla la ubicación del centroide, el cual se representa a través de un cuadrado de color azul, que cambia a verde cuando se encuentra dentro de una región segmentada, esto con el fin de depurar la información recibida, ignorar segmentos muy pequeños.

Figura 27. Módulo para el cálculo y muestra del centroide



Fuente: Este trabajo.

Con el fin de permitir al software de aplicación y al procesador acceder a la información de las coordenadas del centroide, el núcleo incluye un esclavo Avalon MM, que controla los registros que almacenan las coordenadas calculadas por el módulo, de igual forma, permite al software modificar las coordenadas en tiempo real.

Antes de pasar las coordenadas al registro que leerá el procesador, se aplica un filtrado temporal usando el promedio anterior y el presente con la siguiente fórmula:

$$□_□ = (1 - □)□_□ + □□_{□-1}$$

En donde:

$□_□$: Promedio actual
$□_□$: Promedio filtrado actual
$□_{□-1}$: Promedio filtrado anterior
Aplicar esta fórmula permitió suavizar un poco el ruido en el movimiento del centroide, sin embargo, se requería un mejor filtrado del mismo, por lo cual se optó implementar un filtro de Kalman.

Figura 28. Centroide de una región correspondiente a un rostro.

Fuente: Este trabajo.

2.4. DISEÑO DE SOFTWARE DE APLICACIÓN

2.4.1. Creación del BSP.

La programación de la aplicación que se ejecutara en software a través del procesador Nios II, este procesador se encuentra implementado dentro de la FPGA, al igual que los componentes personalizados. Para permitir que el software creado para el procesador pueda acceder a los dispositivos de hardware y componentes de la FPGA, la herramienta de desarrollo Nios II requiere de un paquete de soporte de tarjeta o BSP, que abarca todos los controladores y librerías base para el programa.

La herramienta Nios II SBT puede generar el BSP a través del archivo de descripción del sistema generado por Qsys, este proceso se invoca al generar un nuevo proyecto de programa en la herramienta Nios II, y es necesario generarla cada vez que se realicen cambios sobre el diseño de hardware.

Además del archivo de descripción se requiere especificar las características de las librerías, con el fin de establecer el balance entre consumo de recursos y funcionalidades a implementar, entre las opciones más importantes a configurar se encuentran:

- Sistema operativo: Se configura el BSP para utilizar e implementar el sistema operativo MicroC/OS II, los encabezados y rutinas de inicio cargan las librerías provistas por Micrium, introduciendo el concepto de tareas al código a desarrollar (Tabla 3).

- Soporte de lenguaje: Se establece el soporte para el lenguaje C, la programación orientada a objetos del lenguaje C++ requiere de mayor cantidad de memoria, y los conceptos de la programación orientada a objetos aumenta significativamente la complejidad del programa (Tabla 3).

Tabla 3. Características básicas del BSP

Característica	Valor	Observaciones
Archivo de información SOPC	`system.sopcinfo`	Archivo generado por Qsys que describe la estructura del sistema
Nombre de CPU	`nios2`	Procesador del sistema
Sistema Operativo	Micrium MicroC/OS II	
Archivo BSP	`de2_bsp`	Archivo generado por la aplicación de programación

Fuente: Este trabajo.

- Opciones de la capa de abstracción de hardware (HAL): En las opciones de la HAL se configuran los dispositivos a utilizar, así como el mapeo de dispositivos de entrada y salida a los dispositivos de hardware, de esta forma, se usa la pantalla LCD como salida `stdout` mientras que la entrada se establece a través de la conexión JTAG UART a través de USB, para propósitos de depuración (Tabla 4).

Tabla 4. Mapeo de características de las librerías de C hacia dispositivos del sistema

Característica	Valor	Observaciones
`stderr`	`jtag_uart`	Salida de depuración
`stdin`	`jtag_uart`	Entrada
`stdout`	`lcd_display`	Salida estándar
`sys_clk_timer`	`timer`	Temporizador del sistema
`timestamp_timer`	`timer`	Temporizador de registro

Fuente: Este trabajo.

- Opciones de la memoria de trabajo y de aplicación: Las variables definidas dentro del código fuente y utilizadas por la aplicación diseñada, requieren de un espacio en la memoria, la organización de los espacios de la memoria dependiendo si corresponde a una variable o a las instrucciones del programa compilado se realiza en la configuración del *linker*. La región de memoria correspondiente al *reset*, establece en qué dirección de la memoria el procesador empieza a ejecutar las instrucciones (Tabla 5).

Tabla 5. Mapeo de memoria para el *linker*

Región de memoria del *linker*	Rango de direcciones	Tamaño (bytes)
`flash`	`0x01400020 - 0X017FFFFF`	`4.194.272`

Región de memoria del *linker*	Rango de direcciones	Tamaño (bytes)
reset	0x01400000 - 0X0140001F	32
sdram	0x00300020 - 0x00FFFFFF	8.388.608

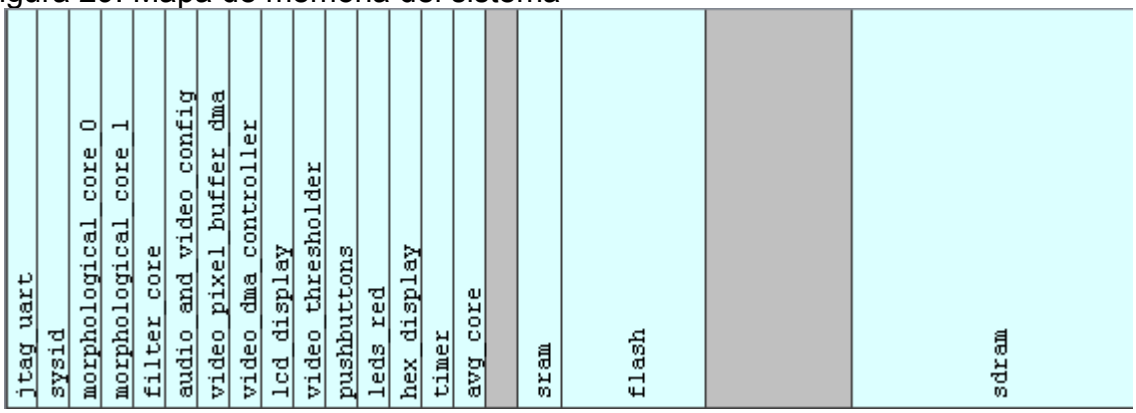
Fuente: Este trabajo.

Tabla 6. Asignación de memoria para los dispositivos

Dispositivo	Rango de direcciones	Tamaño (bytes)
jtag_uart	0x018810E0-0X018810E7	8
sysid	0X018810DS-0x018810DF	8
morpholcgical_core_0	0x018810D0-0x01881007	8
morphological_core_l	0X018810C8-0x018810CF	8
filter_core	0X018810C0-0X018810C7	8
audio_and_video_config	0x018810B0-0x018810BF	16
video_pixel_buffer_dma	0X018810A0-0X018810AF	16
video_dma_controller	0x01881050-0X0188105F	16
lcd_display	0x01881080-0X0188108F	16
video_thresholder	0x01881070-0X0188107F	16
pushbutcons	0x01881060-0x018810€F	16
leds_red	0x01881050-0X0188105F	16
hex_display	0x01881040-0X0188104F	16
timer	0x01881020-0X0188103F	32
avg_core	0x01881000-0x0188101F	32
sram	0x01800000-0X0187FFFF	524288
flash	0x01400000-0X017FFFFF	4154304
sdram	0x00800000-0x00FFFFFF	8388608

Fuente: Este trabajo.

Figura 29. Mapa de memoria del sistema



Fuente: Este trabajo.

- Controladores de dispositivos: Para cada módulo implementado en Qsys se le asigna un controlador. Para el caso del búfer de pixel DMA y el controlador DMA se desarrolla un controlador personalizado, pues estos no están incluidos en el paquete *University Program* como se detalla en la Tabla 7.

Tabla 7. Controladores utilizados en el proyecto de software

Dispositivo	Nombre del modulo	Controlador
audio_and_video_config	altera_up_avalon_audio_and_video_config	Avalon_Audio_and_Video_driver
avg_core	avg_core	avg_core_driver
filter_core	filter_core	none
flash	altera_generic_tristate_controller	altera_ava1on_cfi_flash_driver
hex_display	altera_up_avalon_parallel_port	avalon_parallel_port_driver
jtag_uart	altera_avalon_jtag_uart	altera_avalon_jtag_uart_driver
lcd_display	altera_avalon_lcd_16207	altera_avalon_lcd_16207_driver
leds_red	altera_up_avalon_parallel_port	avalon_parallel_port_driver
morphological_core_0	morpholcgical_core	morpholcgical_core_driver
morphological_core_1	morphological_core	morphological_core_driver
nios2	altera_nios2_qsys	altera_nios2_qsys_ucosii_driver
pushbuttons	altera_up_avalon_parallel_port	avalon_parallel_port_driver
sdram	altera_avalon_new_sdram_controller	none
sram	altera_up_avalon_sram	none
sysid	altera_avalon_sysid_qsys	altera_avalon_sysid_qsys_driver
timer	altera_avalon_timer	altera_avalon_timer_driver
video_dma_controller	altera_up_avalon_video_dma_controller	none
video_pixel_buffer_dma	altera_up_avalon_video_pixel_buffer_dma	avalon_video_pixel_buffer_dma_driver
video_thresholder	video_thresholder	video_thresholder

Fuente: Este trabajo.

- Características y paquetes de software adicionales: Aunque Altera presta soporte para características adicionales como un sistema de archivos compacto o la pila TCP/IP INiche Stack, se implementa únicamente el controlador del *display* hexadecimal (`altera_quad_seven_seg`), este presta una serie de funciones sencillas que simplifica el control de este *display*.

Después de configurar estas características, la aplicación genera los archivos de código necesarios para el desarrollo de aplicaciones. Los encabezados necesarios para interactuar con la HAL se encuentran el archivo `system.h`.

El archivo `system.h` contiene las direcciones de memoria mapeadas y que se conectan al procesador a través del bus Avalon MM. Además, la HAL incluye funciones que permiten leer o escribir en estas direcciones de memoria. Acceder a los dispositivos como si fueran elementos de memoria permite aprovechar la aritmética de punteros del lenguaje C/C++ y lograr así el desarrollo de aplicaciones de una forma flexible.

Las funciones de escritura y lectura provistas por la HAL no son la única forma de interactuar con los dispositivos, ésta también implementa el concepto de descriptores de archivo utilizado por el lenguaje C/C++. Esto permite leer o escribir los dispositivos con las mismas funciones que se utilizan para interactuar con archivos[53].

La mayoría de los dispositivos *University Program* implementados en Qsys proveen los archivos requeridos para interactuar con la HAL. Sin embargo, algunos de estos requieren de la creación y desarrollo de un controlador, siguiendo los estándares y métodos descritos en la documentación provista por el fabricante[54].

2.4.2. Implementación y uso del sistema operativo en tiempo real MicroC/OS II.

La creación del programa que ejecuta el procesador Nios II se realiza programando las instrucciones que serán compiladas y ejecutadas por el procesador. La estructura de la aplicación a implementar y las tareas que realizara el programa puede ser diseñada de varias formas.

La forma más sencilla de estructurar el programa es conocida como *Superloop* en donde todas las tareas del sistema se encuentran dentro de un ciclo infinito que se ejecuta continuamente. Este enfoque, aunque fácil de implementar, se vuelve difícil de manejar cuando el tamaño de la aplicación y la complejidad de las tareas crece, mantener la sincronización de las tareas y llevar el control de las variables del programa puede resultar en un programa susceptible a errores. Un ejemplo en seudocódigo de esta arquitectura se muestra a continuación:

```
Function Función_Principal()
{
  Inicialización();
  Ejecutar_siempre
  {
    Verificar();
    Calcular();
    Salida();
  }
}
```

[53] ALTERA CORPORATION. Nios II Classic Software Developer's Handbook. San Jose, CA: Altera Corporation (2015).
[54] ALTERA CORPORATION. Qsys System Design Tutorial. San Jose, CA: Altera Corporation (may, 2015).

Un enfoque alternativo para la estructuración es la arquitectura multitarea, a través de un sistema operativo en tiempo real RTOS, en este caso, se programan múltiples tareas, asignándoles una prioridad, y el sistema operativo será el encargado de administrar la ejecución de estas. Por ejemplo, una tarea puede estar dedicada a mostrar un valor en la pantalla LCD, mientras otra en leer una entrada, de acuerdo a la prioridad establecida, el sistema operativo puede suspender una tarea para ejecutar otra, y resumir la tarea pausada una vez la tarea ejecutada finalice.

```
Function Tarea_1()
{
  Ejecutar_siempre
  {
    Esperar_Condición();
    Verificar();
    Calcular();
    Salida();
  }
}

Function Tarea_2()
{
  Ejecutar_siempre
  {
    Esperar_Condición();
    Verificar();
    Calcular();
    Salida();
  }
}

Function Función_Principal()
{
  Ejecutar_siempre
  {
    Esperar_Condición_1();
    Llamar_Tarea_1();
    Esperar_Condición_2();
    Llamar_Tarea_2();
  }
}
```

Las herramientas de desarrollo proveen una implementación del sistema operativo MicroC/OS II, el cual implementa una arquitectura multitarea, la aplicación del prototipo implementa el programa dividiéndolo en las tareas mostradas en la Tabla 8.

Tabla 8. Descripción de las tareas implementadas.

Tarea	Descripción
`init_task`	Inicializa los dispositivos, variables globales y ejecuta las tareas de configuración iniciales. Esta tarea se ejecuta una vez, después se suspende y no se ejecuta hasta reiniciar el sistema.
`menu_task`	Gestiona y maneja las acciones e interrupciones de los interruptores para controlar el menú mostrado en la pantalla LCD, esta tarea se ejecuta cuando se genera una interrupción por la activación de los botones *push*.
`kalman_task`	Actualiza y ejecuta un paso del filtro Kalman, obtiene los resultados y los almacena para el siguiente paso del filtro, y los envía a la siguiente tarea.
`update_avg_task`	Tomando los datos del filtro Kalman, actualiza la posición del promedio mostrado en la pantalla.
`count_task`	Esta tarea implementa la lógica necesaria para contar un usuario.
`update_counter_task`	Actualiza los displays siete segmentos con el valor de la cuenta actualizado.

Fuente: Este trabajo.

2.4.3. Diseño y personalización de controladores para dispositivos.

Con el fin de utilizar los componentes diseñados y hacer que se puedan utilizar dentro de la aplicación, se crean paquetes de controladores para los filtros personalizados, en estos se implementaron buses Avalon MM, que permiten controlar registros internos que afectan el comportamiento del módulo. La comunicación entre el procesador y los componentes se realiza escribiendo a una región de memoria establecida durante el diseño del sistema en Qsys y mostrada en la Tabla 6.

Es posible simplificar y abstraer la escritura de estas secciones de memoria a través de software de controlador, para el desarrollo de controladores, Altera provee una serie de librerías estándar conocida como HAL, esta se encuentra descrita en la referencia del procesador Nios II y requiere la creación de funciones y variables que representen los registros asignados y que controlen el hardware a través de funciones en lenguaje C.

2.4.4. Controlador del segmentador de video

El segmentador de video, permite que el valor de los colores de segmentación pueda ser modificados escribiendo en registros internos y comunicados con el procesador a través del bus Avalon MM. Las funciones implementadas se muestran en la Tabla 9.

Tabla 9. Funciones HAL personalizadas para el segmentador de video

Función	Descripción
video_thresholder_read_cb_low(video_thresholder) **video_thresholder_read_cb_high(video_thresholder)** **video_thresholder_read_cr_low(video_thresholder)** **video_thresholder_read_cr_high(video_thresholder)**	Leen los valores de segmentación actual.
video_thresholder_write_cb_low(video_thresholder, data) **video_thresholder_write_cb_high(video_thresholder, data)** **video_thresholder_write_cr_low(video_thresholder, data)** **video_thresholder_write_cr_high(video_thresholder, data)**	Escriben sobre el registro correspondiente modificando los valores del umbral de segmentación en tiempo real.

Fuente: Este trabajo.

2.4.5. Controlador de los filtros

Los filtros incluyen interruptores que modifican el comportamiento de los mismos, el filtro morfológico puede ser alternado entre un filtro de erosión y uno de dilación, mientras que los demás filtros pueden cambiar su estado entre encendido y apagado. Para esto se exponen los registros dentro de los módulos que controlan su funcionamiento a través de un bus Avalon MM conectado al procesador.

Las funciones de control creadas para estos filtros se muestran en la Tabla 10, estas se definen en los archivos de C, `morphological_core` y `filter_core` y se incluyen dentro del BSP del sistema.

Tabla 10. Funciones para el control de los filtros

Función	Descripción
morphological_core_data	Lee el estado actual del filtro morfológico y el modo en el que se encuentra.
morphological_core_enabled	Activa o desactiva el filtro morfológico|
morphological_core_mode	Selecciona la operación a realizar: dilatación o erosión
filter_core_data	Lee el estado actual del filtro de mediana

Función	Descripción
`filter_core_enabled`	Habilita o deshabilita el filtro de mediana

Fuente: Este trabajo.

2.4.6. Controlador del bloque de promedio

El módulo de cálculo de promedio obtiene el centroide de la región segmentada, las funciones de este módulo tienen como objetivo mejorar el cálculo del centroide y su movimiento para la operación de seguimiento utilizando un filtro de Kalman implementado en software. Para que las aplicaciones del procesador puedan acceder a los datos en bruto se utilizan las funciones de lectura. Estas funciones se incluyen dentro de los archivos de C `avg_core`.

Tabla 11 Funciones HAL personalizadas para el calculador del promedio

Función	Descripción
`avg_core_read_xpos_data_raw(avg_core)` `avg_core_read_ypos_data_raw(avg_core)`	Lee los datos del promedio en bruto obtenidos directamente del bloque.
`avg_core_write_xpos_data(avg_core, data)` `avg_core_write_ypos_data(avg_core, data)`	Escribe los valores de posición del cuadro que representa el promedio en la imagen.

Fuente: Este trabajo.

Una vez los datos en bruto han sido procesados, las funciones de escritura devuelven los datos mejorados al bloque de promedio, en donde utilizando los datos de posición, el centroide es dibujado en la pantalla, y su seguimiento y análisis es realizado por la tarea de conteo.

2.4.7. Implementación de la memoria Flash.

Dada la migración realizada por Altera hacia la creación de sistemas con Qsys en reemplazo de la herramienta SOPC, se modificaron varios componentes incluidos con el *University Program*[55], aunque algunos funcionaban correctamente con el proceso de migración incluido en Qsys, otros componentes requirieron de ajustes,

[55] ALTERA CORPORATION. Altera University Program Flash Memory IP Core. San Jose, CA: Altera Corporation (2013).

para lo cual se usaron los pasos sugeridos en la guía de migración de SoPC a Qsys de Altera[56].

El bloque de memoria flash incluido en el *University Program* presenta una forma sencilla para escribir en el dispositivo como si se tratara de una memoria RAM, y provee un registro adicional para borrar la dirección deseada, sin embargo, este enfoque no es útil para el uso de la memoria flash como memoria del programa, ya que el *bootloader* incluido en las herramientas de Nios II, requieren de un acceso directo y de bajo nivel a la memoria.

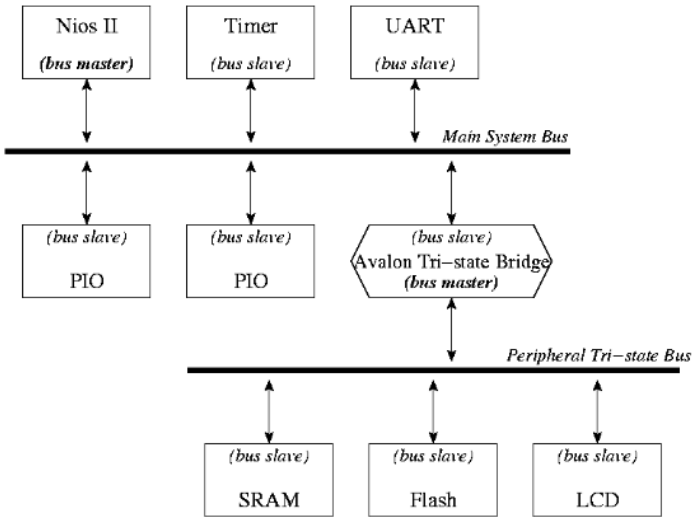
La memoria Flash S29AL032D utiliza lógica tri-estado (nivel alto, nivel bajo y alta impedancia). Con el fin de poder interactuar con el bus binario del resto de componentes del sistema se requiere el uso de un puente tri-estado Avalon como se muestra en la Figura 30. Este componente sirve de traductor entre los componentes del bus binario incluido el procesador Nios II y componentes tri-estado como la memoria Flash[57]. La configuración de este componente no presenta mayores dificultades pues se incluye como parte estándar del bus Avalon.

En cambio, para la configuración de la memoria Flash se requiere conocer las características de temporización y valores específicos de la memoria Flash S29AL032D[58]. De esta forma se provee una interfaz de bajo nivel, que puede ser usada por el programador Flash de Nios II (*Flash Programmer*), el cual puede almacenar la aplicación desarrollada y mantenerla, aunque no exista alimentación de corriente.

---

[56] ALTERA CORPORATION. SOPC Builder to Qsys Migration Opening an SOPC Builder System in Qsys SOPC Builder to Qsys Transformations. San Jose, CA: Altera Corporation (may, 2011).
[57] HAMBLEN, et al. Op. cit.
[58] Qsys における オフチップ ・ メモリ ・ インタフェースの接続方法. (2015).

Figura 30. Esquema del puente tri-estado Avalon (*Avalon Tri-state Bridge*)



Fuente: HAMBLEN, James O., HALL, Tyson S.; FURMAN, Michael D. Rapid Prototyping of Digital Systems. Springer (2008).

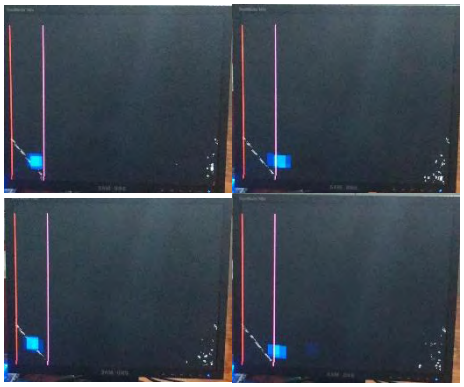
El proceso realizado por el *Flash Programmer* consiste en la implementación de un *bootloader* dentro de la memoria Flash, un programa básico cuya función es copiar el programa en la memoria SDRAM y ejecutar el programa. De esta forma, el programa permanece en la memoria flash no volátil y puede cargarse en la memoria volátil una vez se haya encendido el dispositivo.

Para asegurar que el procesador inicie el *bootloader* desde la memoria flash cada vez que se encendido, es necesario hacer que el vector *reset* del procesador (la dirección en la que el procesador inicia su ejecución) apunte al inicio de la memoria Flash.

2.4.8. Filtro de Kalman

Durante las pruebas iniciales del prototipo, se observó que el promedio presentaba un ruido en su movimiento que afectaba el rastreo y detección como se muestra en la Figura 31. Movimiento errático del centroide.. Este movimiento errático se producía debido a las variaciones de la imagen causadas por el segmentador. Para filtrar este ruido en el movimiento se optó por la utilización de un filtro de Kalman a ser implementado en software.

Figura 31. Movimiento errático del centroide.



Fuente: Este trabajo.

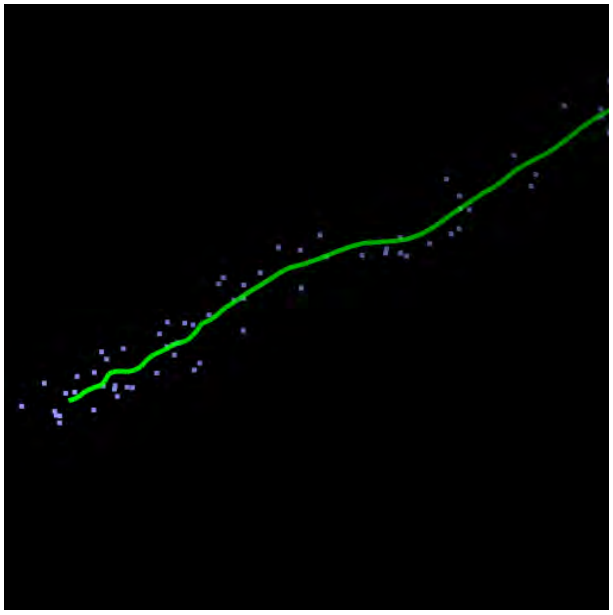
El filtro de Kalman consiste en un conjunto de ecuaciones matemáticas que proveen una solución recursiva óptima, por el método de mínimos cuadrados. La meta de esta solución consiste en calcular un estimador lineal, insesgado y óptimo del estado de un sistema en t con base en la información disponible en $\square - 1$, y actualizar, con la información adicional disponible en $\square$, dichas estimaciones.

El filtro se desempeña suponiendo que el sistema puede ser descrito a través de un modelo estocástico lineal, en donde el error asociado tanto al sistema como a la información adicional que se incorpora en el mismo tiene una distribución normal con media cero y varianza determinada.

El filtro de Kalman es el principal algoritmo para estimar sistemas dinámicos representados en la forma de estado-espacio En esta representación el sistema es descrito por un conjunto de variables denominadas de estado. El estado contiene toda la información relativa al sistema a un cierto punto en el tiempo. Esta información debe permitir la inferencia del comportamiento pasado del sistema, con el objetivo de predecir su comportamiento futuro.

La implementación de este filtro se realiza en software del procesador Nios II, utilizando la librería de software libre TinyEKF. Esta librería implementa las operaciones básicas de algebra lineal, las ecuaciones del filtro Kalman y las expone al software del programa a través de una estructura capaz de almacenar todos los datos del modelo.

Figura 32. Funcionamiento del filtro Kalman sobre una muestra ruidosa



Fuente: Este trabajo.

El modelo a utilizar corresponde a un simple modelo de movimiento en x, y, que corresponde a la localización del promedio obtenida por el nucleo `avg_core`. Por lo general, la salida de los datos en bruto presenta bastante ruido en su movimiento, como se observa en la Figura 32. Funcionamiento del filtro Kalman sobre una muestra ruidosa. El filtro Kalman realiza un "suavizado" de este movimiento, realizando una media entre el valor predicho y el valor medido, reduciendo en cada paso la varianza entre el valor filtrado y los datos medidos.

La librería TinyEKF requiere de un filtro Kalman previamente diseñado, para esto establecemos los parámetros necesarios a continuación:

En primer lugar seleccionamos las variables de estado, en nuestro caso, la posición en x y en y y su respectiva velocidad.

$$\square = [\square \quad \square_{\square} \quad y \quad v_y]^{\square}$$

Luego diseñamos la función de transición, esta representa el modelo matemático del proceso, para el caso del movimiento, la posición en el futuro corresponde a la posición presente mas la velocidad por el diferencial del tiempo:

$$\square = 1\square + \Delta\square\square_{\square} + 0\square + 0\square_{\square}$$

$$\square_{\square} = 0\square + 1\square_{\square} + 0\square + 0\square_{\square}$$

$$\square = 0\square + 0\square_{\square} + 1\square + \Delta\square\square_{\square}$$

$$\square_{\square} = 0\square + 0\square_{\square} + 0\square + 1\square_{\square}$$

La representación en espacio de estados de estas ecuaciones es:

$$\begin{bmatrix} 1 & \Delta\square & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta\square \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

La matriz Q representa la varianza del ruido del proceso, su modificación hace que el filtro siga más a la predicción o a la medición. La librería recibe un parámetro Qxy, el cual genera la matriz Q para un proceso de movimiento en 2 dimensiones.

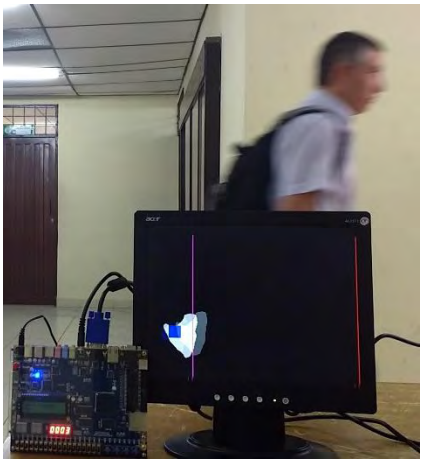
Al elegir un valor de Qxy de 0.1 se logra que el movimiento del centroide sea fluido y siga al sujeto de conteo ignorando el ruido de objetos ajenos al conteo.

2.4.9. Conteo

El centroide filtrado se envía a la tarea de conteo `count_task`, esta tarea analiza la posición y velocidad del centroide para establecer si ha superado el umbral establecido en el menú. Si supera el valor en x del umbral y su dirección es hacia este, se establece que el sujeto ha atravesado el umbral limite.

Con el fin de evitar falsos positivos y que el prototipo cuente a sujetos que salgan del recinto se establece un segundo umbral, ubicado en el centro del área de detección. Este segundo umbral mejora sustancialmente la detección y solo tiene en cuenta los sujetos que atraviesen completamente el área de detección en dirección de entrada como en el caso de la Figura 33. Sistema mostrando el umbral de salida siendo atravesado en sentido contrario..

Figura 33. Sistema mostrando el umbral de salida siendo atravesado en sentido contrario.



Fuente: Este trabajo

3. RESULTADOS

3.1. PRUEBAS EN SOFTWARE

Con el fin de estudiar y familiarizarse con los algoritmos apropiados para la detección facial a través del color de piel, estos se implementan en MATLAB ®, y se prueban con imágenes representativas, para así ajustar los parámetros y proceder luego a la implementación de estos en hardware.

3.1.1. Base de datos ColorFERET.

A mediados de los años 90, el Departamento de Defensa estadounidense (DoD) a través de la Agencia para la investigación de productos avanzados (DARPA) crearon la base de datos FERET o Tecnología de Reconocimiento Facial, con el fin de probar y evaluar algoritmos de reconocimiento facial utilizando procedimientos y pruebas estandarizadas[59].

La base de datos es distribuida por el NIST o Instituto Nacional de Estándares y Tecnología, y posee alrededor de 8,5 GBytes de imágenes de rostros, etiquetadas y clasificadas por raza, género, y posición. Se utiliza esta base de datos por la variedad de muestras que incluye permitiendo probar los algoritmos con una gran cantidad de datos.

Las imágenes se encuentran formato PPM o mapa de pixeles portable, su principal ventaja es su facilidad de lectura, a cambio de ser ineficiente en el uso del espacio, sin embargo, dado el objetivo de estas imágenes, este sacrificio es comprensible.

Las características del sujeto de la imagen se describen en archivos de texto plano. Con el objetivo de realizar una selección aleatoria y significativa de sujetos, se realizó una aplicación en MATLAB que lee los archivos de texto plano y crea una estructura que contiene la información requerida. A continuación, se realiza la selección a partir de las características deseadas (raza, sexo, posición) y por último escribe la imagen a una variable para poder así trabajar con ella.

3.1.2. Segmentador de imágenes.

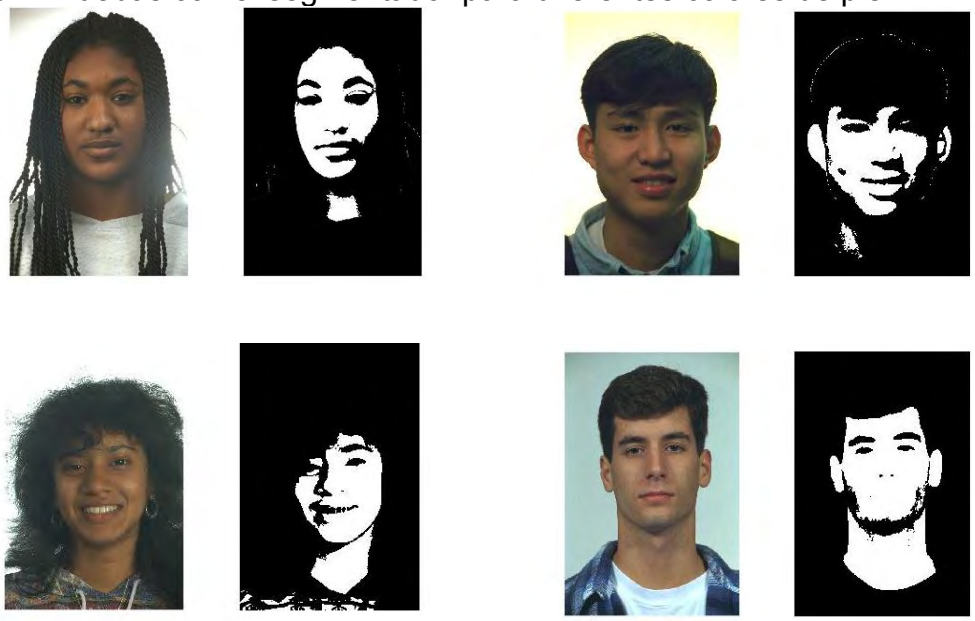
El segmentador facial descrito en la sección 2.3.1, se implementa en MATLAB a través de la función `thresholder.m`, la cual toma como entrada una matriz tridimensional en donde la tercera dimensión representa los canales de la imagen y arroja una imagen binaria, en donde el 1 representa un pixel de piel y el 0 lo contrario. Para probar el algoritmo se seleccionaron varias imágenes de la base de datos ColorFERET de distintas razas y colores de piel, los resultados en algunas

---

[59] Face Recognition Technology (FERET). (Enero 25, 2011).

de estas se pueden observar en la Figura 34. En ellas se observa que el segmentador arroja muy buenos resultados y es capaz de detectar la piel en todas las imágenes seleccionadas.

Figura 34. Pruebas con el segmentador para diferentes colores de piel



Fuente: Este trabajo

3.1.3. Filtro morfológico.

En la práctica existen muchos otros objetos con color similar al color de piel, muchos de estos falsos positivos se encuentran presentes en el resultado de la segmentación. Aplicar filtrado morfológico, como erosión o dilatación, reduce el ruido de fondo y llenaría los pixeles faltantes de las regiones detectadas como se muestra en la Figura 35.

MATLAB implementa funciones morfológicas básicas, como `imerode` o `imdilate`, las cuales permiten el uso de cualquier elemento estructurante. Con el fin de hacer que la simulación en la aplicación sea similar y tenga en cuenta las limitaciones de hardware, se utiliza un elemento estructurante cuadrado de 7 x 7.

Figura 35. Aplicación del filtro de apertura (erosión y dilación)

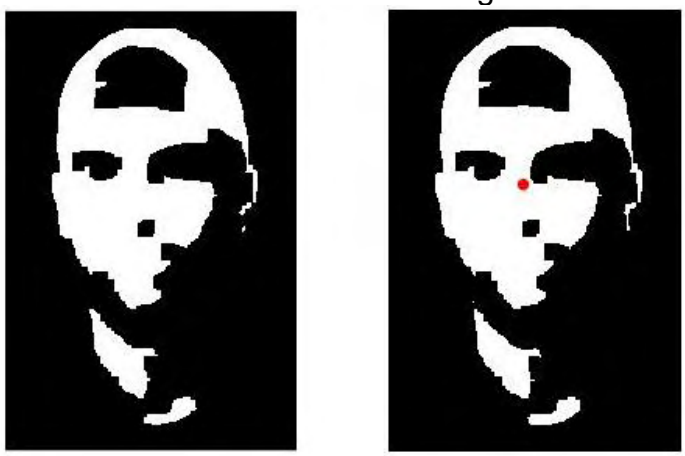
Fuente: Este trabajo

3.1.4. Cálculo del centroide.

Con el fin de establecer la localización del rostro, el centroide de la imagen se calcula promediando la suma de las coordenadas en X y Y de los pixeles correspondientes a piel, este resultado se gráfica y representa con un punto rojo como se observa en la Figura 36.

También es posible utilizar la función `regionprops`, la cual separa cada región conectada y asigna un centroide a cada una, aunque efectiva, su implementación en FPGA es compleja y tomaría muchos más recursos de los disponibles.

Figura 36. Ubicación del centroide dentro de la imagen



Fuente: Este trabajo

3.1. PRUEBAS DE HARDWARE

Con el propósito de realizar las pruebas de hardware implementadas en la tarjeta de desarrollo Altera DE2, se conecta una cámara para la captura de video a la

entrada análoga de la tarjeta y un monitor a la salida de video VGA de la misma. Una vez implementados los módulos necesarios para la funcionalidad del puerto VGA se puede analizar los resultados del procesamiento en tiempo real en el monitor. El esquema general del montaje se muestra en la Figura 37.

Figura 37. Esquema de la instalación para realizar las pruebas

Fuente: Este trabajo

Durante el desarrollo del proyecto se utilizó un total de 4 cámaras diferentes con el propósito de encontrar el dispositivo que mejor se adapte al sistema y al objetivo final del proyecto.

Inicialmente se utilizó una video cámara Sony CCD-TRV58 Figura 38, con la cual se realizaron pruebas de segmentación, filtrado y cálculo de centroide de la imagen. No se realizaron pruebas de seguimiento con esta cámara, pues el dispositivo se encontraba muy expuesto a cambios de color debido a la sensibilidad a los cambios de luz, lo cual afecta considerablemente los parámetros del sistema para esta función.

Finalmente, se optó por adquirir una cámara de seguridad especializada para video vigilancia, pues estás se encuentran expuestas a difíciles condiciones tanto ambientales como de luz. Con el propósito de encontrar el dispositivo que mejor se ajuste a las necesidades del sistema se adquirieron 3 tipos de cámaras disponibles en el mercado Figura 38, realizando pruebas con cada una de ellas.

Figura 38: Cámaras utilizadas para la captura de video

a. Sony CCD-TRV58

b. CCI-MPA000

c. GVS AHDDP720PF28L24

d. ONVISION ONDM72V2812L36HD

Fuente:

La cámara ONVISION ONDM72V2812L36HD fue seleccionada para la captura de video del sistema pues no presenta una variación considerable en los colores de la imagen capturada a pesar de los cambios de luz del ambiente. Sus características más importantes se resumen en la Tabla 12.

Tabla 12. Características cámara video vigilancia

ITEM	**DESCRIPCIÓN**
Marca	ONVISION
Modelo	ONDM72V2812L36HD
Tipo	Domo
Sensor	CMOS 1.0 Megapíxeles
Resolución	1280x720

ITEM	DESCRIPCIÓN
Taza de video	50 fps, 60 fps
Iluminación mínima	0,01 lux
Balance de blancos	Automático
Salida de video	AHD (*Analog High Definition*)
Voltaje	12 V – DC
Corriente	400 mA - CD
Lente	2,8 – 12 mm
Dimensiones	118mm x 96,5mm
Peso	700 gr

Fuente: ONVISION. ONDM72V2812L36HD User Guide. (2015).

Figura 39. Cámara ONVISION modelo ONDM72V2812L36HD

Fuente: CCD Camera Manual - CCI-MPA000. (2016).

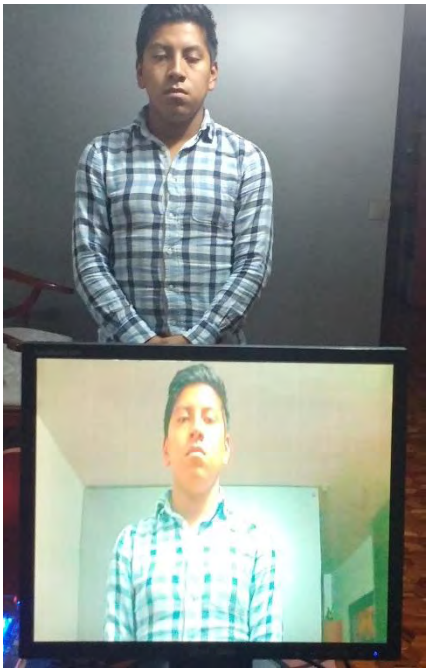
3.1.1. Desentrelazador.

Definir desentrelazador. El algoritmo de desentrelazado implementado en la FPGA es el algoritmo Bob, el cual reduce la resolución. Dadas las necesidades del proyecto, la reducción en la resolución no afecta el cumplimiento de los objetivos, esto junto a los bajos requerimientos en su implementación hacen que sea el método de desentrelazado ideal para el prototipo.

3.1.2. Filtro gaussiano.

La aplicación del filtro gaussiano suaviza la imagen y elimina el ruido que puede afectar la detección, en especial el ruido ISO de la cámara el cual se presenta en condiciones de luz adversas, siendo equivalente a un filtro espacial pasa bajos. Usando todos los parámetros del filtro descrito en 2.3.3 se procesa y filtra la imagen en tiempo real.

Figura 40. Resultado después de aplicar el filtro gaussiano.



Este trabajo

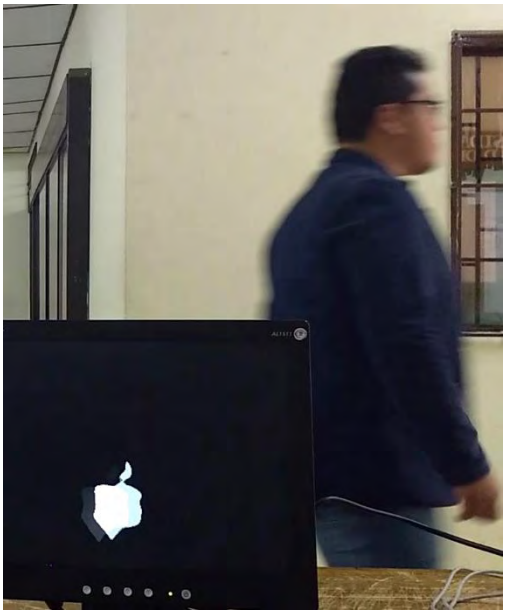
La aplicación del filtro en tiempo real utilizando el núcleo descrito por Serfa Juan[60] ofrece buenos resultados, el ruido presentado en la cámara se reduce considerablemente al medirlo de manera cualitativa; esta reducción mejora la precisión y eficiencia de los filtros subsecuentes.

3.1.3. Segmentador.

El segmentador implementado y descrito anteriormente es capaz de detectar los colores correspondientes a la piel, y generar una imagen binaria en tiempo real, la cual se muestra en la Figura 41. Las pruebas realizadas en condiciones de luz adecuadas, es decir, el sujeto está expuesto a una fuente de luz de más de 1000 lumens. Lo suficiente para que la cámara de video puede reproducir los colores exitosamente.

[60] SERFA JUAN, et al. Op. cit.

Figura 41. Resultado del segmentador de imagen

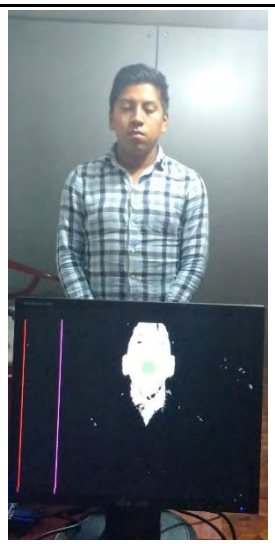


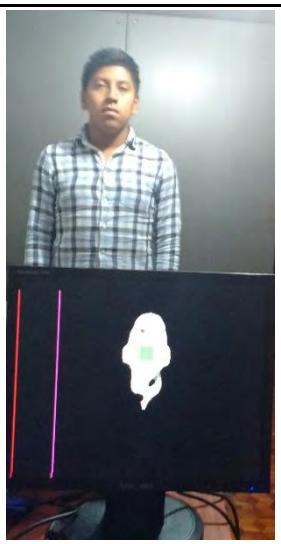
Este trabajo

La imagen binaria mostrada representa los valores lógicos de cada pixel, 1 cuando se ha clasificado el pixel como un pixel con color de piel y 0 en caso contrario.

3.1.4. Filtro de mediana

El filtro de mediana es capaz de eliminar el ruido sal y pimienta generado por el segmentador al detectar pequeños puntos cuyo color corresponde al de la piel pero no son parte del sujeto de detección. Este filtro mejora sustancialmente la calidad de la segmentación por lo que su uso es esencial durante la ejecución del prototipo.



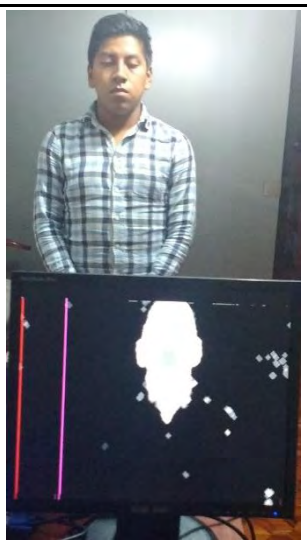


Original	Filtrado

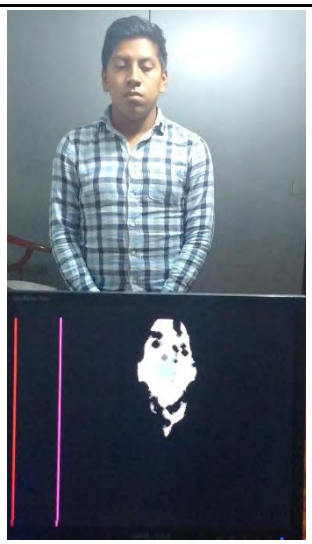
3.1.5. Filtro morfológico.

El aplicar el filtro de apertura y cerramiento mejora la imagen eliminando el ruido y las regiones pequeñas. Así se crea una imagen con mayor consistencia, lo que facilita el análisis con el fin de determinar la ubicación del rostro.

Figura 42. Resultado del filtrado morfológico



Dilación



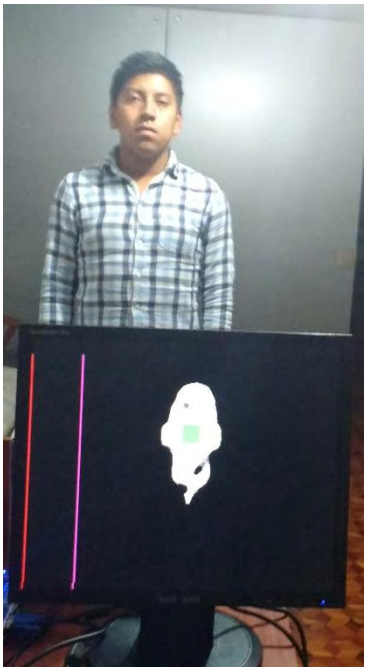
Erosión

Este trabajo

3.1.6. Cálculo del centroide.

El centroide implementa el método del promedio descrito en 2.3.7, tomando la imagen filtrada es capaz de detectar y marcar en un recuadro rojo y en tiempo real, la ubicación del centro de la imagen, el rastreo se realiza de forma fluida y rápida, respondiendo al movimiento del rostro de forma exitosa. El componente es capaz de modificar la memoria en tiempo real indicando las coordenadas del centroide. De esta forma el módulo de rastreo puede determinar la posición del rostro.

Figura 43. Cálculo del centroide de la imagen



Este trabajo

3.1.7. Rastreo.

La aplicación de rastreo lee los valores del centroide en la memoria y determina si se encuentra en la región que se determine como “borde”, y modifica el contador interno, este se muestra en el *display* 7 segmentos de la FPGA y presenta al usuario con la cuenta de personas que han atravesado la cámara.

Las coordenadas de los centroides se almacenan en la memoria de la aplicación y son procesados en software, esto con el objetivo de evitar que centroides ajenos al del rostro detectado afecten el resultado, esto se realiza teniendo en cuenta únicamente el punto que este más cercano al punto detectado en el cuadro anterior y filtrando los puntos alejados.

3.2. PRUEBAS DEL PROTOTIPO

3.2.1. Metodología de la Prueba

Para la prueba del prototipo de hardware se instaló el dispositivo en la recepción de una oficina administrativa y en el pasillo de los laboratorios de ingeniería electronica, capturando el video en un dispositivo de grabación y enviándolo a la tarjeta de desarrollo a través del puerto de video análogo.

La cantidad de personas en el video grabado, se contó manualmente, con el fin de establecer la base de comparación para el dispositivo, la prueba se realizó en

horas de la mañana con condiciones de luz favorables para la reproducción de color.

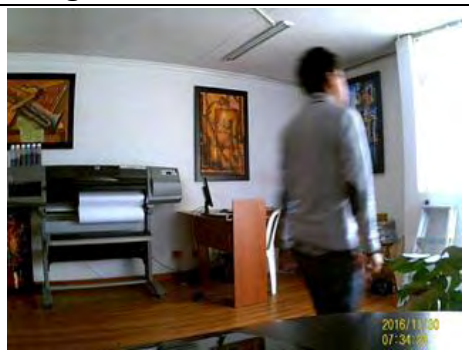
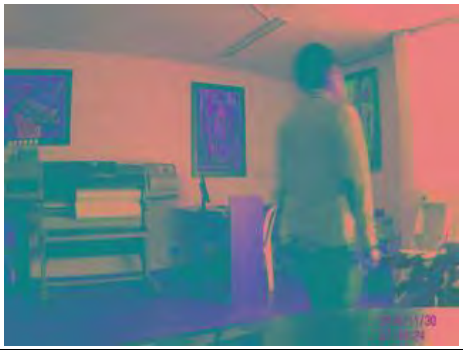
3.2.2. Resultados

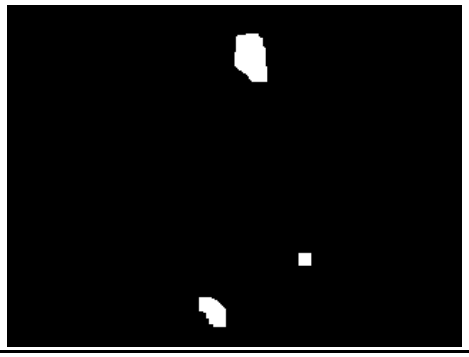
Los resultados de la captura muestran el funcionamiento de los diferentes procesos aplicados a la imagen en tiempo real dentro de la FPGA. En primer lugar, la conversión de espacio de color al espacio YCbCr, objeto de la segmentación, facilita la detección de regiones de color correspondiente a la piel del sujeto.

El filtrado de imagen se aplica a la imagen fuente, para eliminar fuentes de ruido de la señal, particularmente las causadas por los sensores CCD. Este filtro gaussiano de mediana fuerza se ajusta con el fin de evitar la eliminación de información de color y mantener las regiones de interés de la imagen.

La segmentación facial es ayudada por la eliminación de elementos estáticos tomando como referencia el primer cuadro, es por esto que se debe reiniciar el prototipo a través del botón *reset*, una vez sea establecido en la ubicación estable y sin sujetos atravesando el campo de visión de la cámara.

Tabla 13. Resultados obtenidos con una captura de ejemplo.

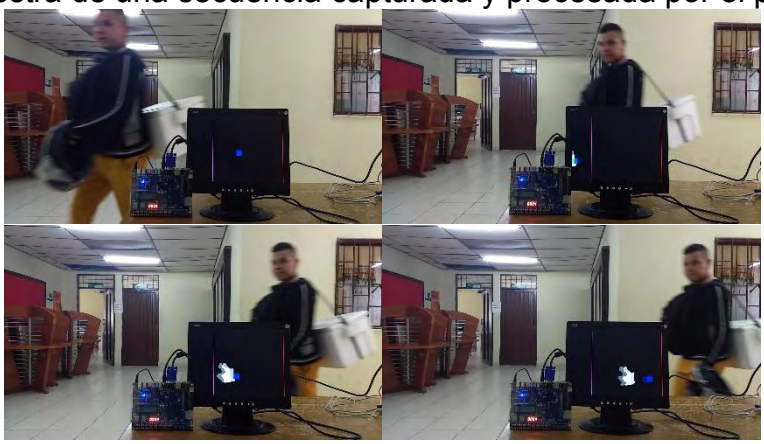
Descripción	Imagen
La imagen capturada a color, se observa al individuo atravesando el campo de visión de la cámara.	
La imagen se convierte al espacio de color YCbCr en la FPGA, se muestra una visualización de la imagen en este espacio.	

Descripción	Imagen
La imagen después de realizar la segmentación de color, se observa el área de piel correspondiente al rostro y los brazos.	
La imagen después de realizarse el filtro morfológico, se observa que la región de interés a un mayor tamaño facilitando el rastreo.	

Fuente: Este trabajo

Por último, un filtrado temporal permite deshacerse de las regiones que son detectadas en cuadros de manera esporádica y que suelen corresponder a cambios de brillo instantáneos de la imagen.

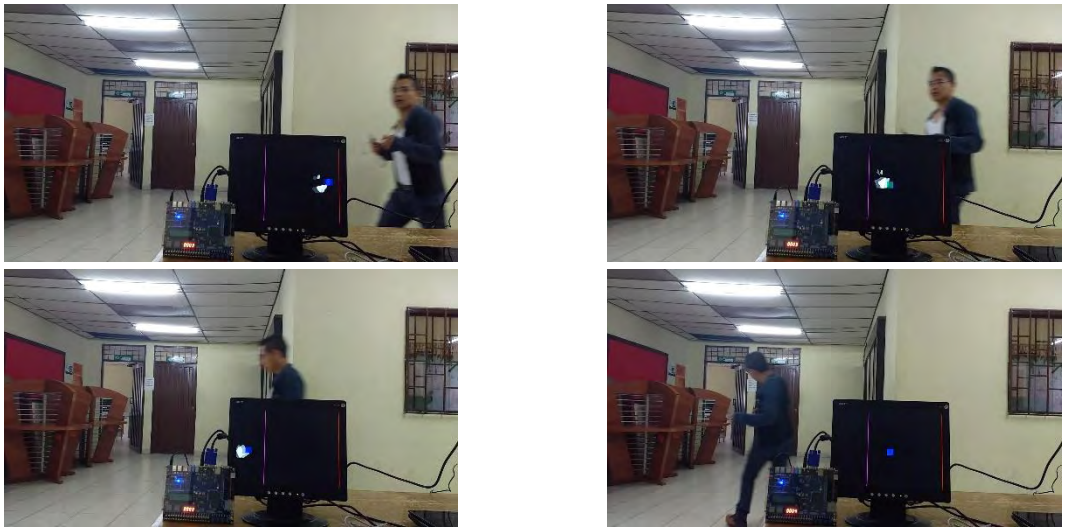
Figura 44. Muestra de una secuencia capturada y procesada por el prototipo



Fuente: Este trabajo

Secuencias posteriores permiten observar la detección realizada por la FPGA, contando sujetos en movimiento ignorando sujetos que no atraviesen el umbral establecido como se observa en la Figura 45.

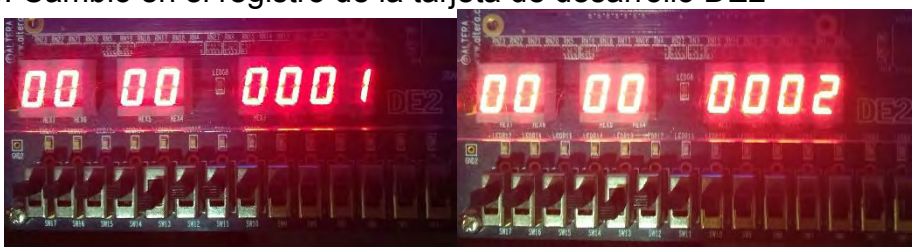
Figura 45. Pruebas sucesivas y segmentación.



Fuente: Este trabajo

La Figura 46 muestra una secuencia que fue procesada por el prototipo, en este caso el dispositivo contó exitosamente la entrada adicionando al registro y mostrándolo en el *display* 7 segmentos.

Figura 46. Cambio en el registro de la tarjeta de desarrollo DE2



Fuente: Este trabajo

Pruebas sucesivas mostraron resultados satisfactorios, el dispositivo contó las personas que ingresaron en la oficina, ignorando fuentes de ruido y elementos del fondo que interferían en un principio con la detección.

Los resultados de las pruebas se muestran en las Tabla 14 y 15, se utilizó iluminación diurna en horas de la tarde de 2 a 4 PM y cada muestra duro 15 minutos de conteo.

Tabla 14. Tabla de resultados obtenida en los laboratorios.

Muestra	**Valor contado por el prototipo**	**Valor de personas que ingresaron**	**Diferencia**
1	17	16	-16
2	12	14	-10

Muestra	Valor contado por el prototipo	Valor de personas que ingresaron	Diferencia
3	10	9	-7
4	7	7	-3
5	19	17	-14
6	21	24	-15

Fuente: Este trabajo

Tabla 15. Tabla de resultados obtenida en una oficina.

Muestra	Valor contado por el prototipo	Valor de personas que ingresaron	Diferencia
1	7	7	0
2	5	5	0
3	9	10	-1
4	7	7	0
5	12	12	0
6	4	4	0

Fuente: Este trabajo

En los casos que el prototipo contó menos personas como por ejemplo la muestra 6 de la Tabla 14, las personas ingresaron juntas, el prototipo no está preparado para estos casos por lo que solo cuenta una sola persona.

Dado que el segmentador es sensible al color, la presencia de buena iluminación es fundamental, los mejores resultados se obtienen en condiciones de luz diurna o luz artificial blanca generada por una fuente superior 1000 lumens. Así mismo algunos objetos pueden interferir con la detección al encontrarse dentro del rango de color de la piel.

Cuando los objetos que no corresponden a piel no son una parte significativa de la imagen, el filtro de Kalman puede ignorarlos debido que este prioriza el movimiento del rastreador y así no afecta el conteo final.

4. CONCLUSIONES

El estudio e implementación previa de los algoritmos en MATLAB ayuda en la compresión de su funcionamiento, esto con el fin de establecer las partes que pueden beneficiarse con la implementación en hardware y lograr que actúen en un sistema en tiempo real.

Requerir el procesamiento en tiempo real hace que se prioricen los algoritmos para la velocidad sobre precisión, uso de recursos y eficiencia energética. Dado que la información procesada corresponde a video, se requiere de una gran capacidad de volumen de datos o *throughput* por parte de los componentes que se creen y se utilicen en la tarjeta de desarrollo.

Antes de realizar cualquier implementación de un algoritmo es necesario estudiar detalladamente las limitaciones de la FPGA, ya que no todos los algoritmos pueden ser implementados, o algunos requieren DSPs personalizados cuya implementación puede sobrepasar las capacidades provistas por el equipo.

El método de segmentación facial utilizando el espacio de color YCbCr ofrece buenos resultados para equipos con recursos limitados gracias a su facilidad de implementación. Múltiples investigaciones han perfeccionado el algoritmo utilizando la menor cantidad de recursos posibles. Aunque otros espacios de color como CIELab ofrecen mejores resultados, la conversión requiere de complejas operaciones sin ofrecer una mejora significativa.

La tasa de falsos positivos en la identificación de piel hace necesario el uso de otras herramientas con el fin de apoyar la detección facial, algunos de estos efectos pueden mitigarse con la iluminación adecuada y la correcta configuración de la cámara para la captura de imágenes.

La implementación de filtros de convolución utilizando el método de desplazamiento de ventana es una herramienta versátil cuya utilidad no se limita a filtros gaussianos o morfológicos, sino que se puede utilizar con detectores de borde, pasa altos y pasa bajo sirviendo de puente entre la implementación y la descripción matemática de las operaciones de convolución de imágenes.

El uso de Qsys como herramienta para la organización de sistemas embebidos junto con el procesador Nios II son una combinación poderosa para la creación rápida de prototipos aprovechando los núcleos IP de terceros y creando los propios con el fin de lograr un sistema con objetivos definidos y utilizando el hardware que resulte estrictamente necesario.

El uso de un procesador permite el desarrollo de aplicaciones con un lenguaje de programación como C, el cual resulta más familiar, facilitando la integración del sistema y evitando el uso exclusivo de lenguajes de descripción de hardware, lo que permite una mayor adaptabilidad y versatilidad del sistema.

5. RECOMENDACIONES Y TRABAJOS FUTUROS

Se sugiere estudiar diferentes técnicas para futuros trabajos en el diseño de un sistema de reconocimiento facial con la tarjeta de desarrollo Altera DE2, como por ejemplo el método Viola-Jones para la detección de objetos o filtros Haar, teniendo en cuenta las características del dispositivo y sus limitaciones.

El uso de tarjetas de desarrollo especializadas en el procesamiento de video abre nuevas puertas a la investigación de algoritmos, su implementación y optimización; nuevos modelos incluyen transmisión HDMI, DSPs especializados e incluso codecs implementados en hardware.

El aprovechamiento de las FPGA con fines pedagógicos para el estudio de sistemas digitales, diseño de sistemas embebidos y arquitectura de procesadores ampliaría las posibilidades de investigación y aplicación para proyectos interdisciplinarios en la Universidad de Nariño.

El uso del puerto paralelo para la integración de componentes personalizados y otras tarjetas de desarrollo amplía las capacidades de la tarjeta y la FPGA, así como sus posibilidades de aplicación.

Aunque la tarjeta posee el hardware necesario para la transmisión y recepción a través de redes de datos, no posee controladores compatibles con la pila de protocolos TCP/IP incluidos en el sistema operativo MicroC/OS II, la creación de dicho controlador abre las puertas a la investigación sobre redes de computadoras abordando temas como la seguridad, encriptación, esteganografía, etc.

BIBLIOGRAFÍA

Face Recognition Technology (FERET). (2011).

Qsys における オフチップ ・ メモリ ・ インタフェースの接続方法. (2015), p. 1-13.

CCD Camera Manual - CCI-MPA000. (2016).

AL-TAIRI, Zaher Hamid, RAHMAT, Rahmita Wirza, IQBAL SARIPAN, M. y SULAIMAN, Puteri Suhaiza. Skin segmentation using YUV and RGB color spaces. En: Journal of Information Processing Systems. jun, 2014. vol. 10, p. 283-299

ALBIOL, Alberto, TORRES, Luis y DELP, Edward J. Optimum color spaces for skin detection. En: Journal of Chemical Information and Modeling. 1989. vol. 53, p. 160

ALTERA CORPORATION. Development and Education Board User Manual. Altera Corporation (2006), p. 23-34.

ALTERA CORPORATION. SOPC Builder to Qsys Migration Opening an SOPC Builder System in Qsys SOPC Builder to Qsys Transformations. San Jose, CA: Altera Corporation (2011).

ALTERA CORPORATION. Altera University Program Flash Memory IP Core. San Jose, CA: Altera Corporation (2013).

ALTERA CORPORATION. Quartus II Handbook Version 13.1 Volume 1: Design and Synthesis. San Jose, CA: Altera Corporation (2013), p. 1820.

ALTERA CORPORATION. Video IP Cores for Altera DE-Series Boards. San Jose, CA: Altera Corporation (2013), p. 1-47.

ALTERA CORPORATION. Avalon Interface Specifications. San Jose, CA: Altera Corporation (2015).

ALTERA CORPORATION. Nios II Classic Processor Reference Guide. San Jose, CA: Altera Corporation (2015).

ALTERA CORPORATION. Nios II Classic Software Developer’s Handbook. San Jose, CA: Altera Corporation (2015).

ALTERA CORPORATION. Qsys System Design Tutorial. San Jose, CA: Altera Corporation (2015), p. 27.

ARUNMOZHI, R. y MOHAN, G. Implementation of Digital Image Morphological Algorithm on FPGA using Hardware Description Languages. En: International Journal of Computer Applications. 2012. vol. 57, p. 1-4

AVIZIENIS, Rimas. Building your First Image Processing ASIC | CS250 Laboratory 2. Autor (2012), p. 10.

BASILIO, Jorge Alberto Marcial, TORRES, Gualberto Aguilar, PÉREZ, Gabriel Sánchez, MEDINA, L. Karina Toscano y MEANA, Héctor M. Pérez. Explicit Image Detection using YCbCr Space Color Model as Skin Detection. En: American Conference on Applied Mathematics (AMERICAN-MATH '11) - 5th WSEAS International Conference on Computer Engineering and Applications (CEA '11) (Puerto Morelos, Mexico). Tertiary Explicit Image Detection using YCbCr Space Color Model as Skin Detection2011. p. 123-128

BENDA, Lukas. Hardware acceleration for image processing. (2008), p. 1-45.

BERBAR, Mohamed A., KELASH, Hamdy M. y KANDEEL, Amany A. Faces and Facial Features Detection in Color Images. En: Geometric Modeling and Imaging – New Trends (GMAI06) (Washington). Tertiary Faces and Facial Features Detection in Color Images: IEEE, 2006. p. 209-214

BIN ABDUL RAHMAN, Nusirwan Anwar, WEI, Kit Chong y SEE, John. RGB-H-CbCr Skin Colour Model for Human Face Detection. En: Proceedings of The MMU International Symposium on Information & Communications Technologies (M2USIC 2006). Tertiary RGB-H-CbCr Skin Colour Model for Human Face Detection2006. p. 90-96

BOVIK, Alan Conrad. Handbook of Image and Video Processing. Academic Press, 2005. ISBN: 978-01-2119-792-6.

COFER, R. C. y HARDING, Benjamin F. Rapid System Prototyping With FPGAs. Newnes, 2005. ISBN: 978-07-5067-866-7.

CHEN, Ching Yi, CHEN, Ching Han y HO, Hsiao Ping. A real-time skin color based human face tracking system using an evolutionary search strategy. En: Journal of Applied Science and Engineering. 2013. vol. 16, p. 249-259

CHU, Pong P. FPGA Prototyping by Verilog Examples: Xilinx Spartan-3 Version. John Wiley & Sons, 2008. ISBN: 978-04-7018-532-2.

CHU, Pong P. Embedded SoPC Design With Nios II Processor and VHDL Examples. Wiley, 2011. ISBN: 978-11-1800-888-1.

CHUCKS, U. VHDL Image Processing. 2011.

DAWSON-HOWE, Kenneth. A Practical Introduction to Computer Vision With OpenCV. Wiley, 2014. ISBN: 978-11-1884-845-6.

GHANNOUM, Anthony. Image Filtering in FPGAs | Teledyne DALSA Imaging Blog. (2012).

GRAF, Hans Peter, COSATTO, Eric, GIBBON, Dave, KOCHEISEN, Michael y PETAJAN, Eric. Multi-modal system for locating heads and faces. En: Proceedings of the Second International Conference on Automatic Face and Gesture Recognition, 1996 (Killington). Tertiary Multi-modal system for locating heads and faces: IEEE, 1996. p. 88-93

GRAF, Hans Peter, CHEN, Tsuhan, PETAJAN, Eric y COSATTO, Eric. Locating faces and facial parts. En: Proceedings of the First International Workshop on Automatic Face and Gesture Recognition. Tertiary Locating faces and facial parts1995. p. 41-46

HAMBLEN, James O., HALL, Tyson S. y FURMAN, Michael D. Rapid Prototyping of Digital Systems. Springer, 2008. ISBN: 978-03-8772-670-0.

HEATH, Steve. Embedded Systems Design. Newnes, 2003.

HU, M., WORRALL, S., SADKA, A. H. y KONDOZ, A. M. A Fast and Efficient Chin Detection Method for 2-D Scalable Face Model Design. En: International Conference on Visual Information Engineering, 2003. VIE 2003. Tertiary A Fast and Efficient Chin Detection Method for 2-D Scalable Face Model Design2003. p. 125-128

KAMAL, Raj. Embedded Systems: Architecture, Programming and Design. New Delhi: Tata McGraw-Hill, 2009.

KAUR, Amanpreet y KRANTHI, B. V. Comparison between YCbCr Color Space and CIELab Color Space for Skin Color Segmentation. En: International Journal of Applied Information Systems. 2012. vol. 3, p. 30-33

KILTS, Steve. Advanced FPGA Design: Architecture, Implementation, and Optimization. Wiley-IEEE Press, 2007. ISBN: 978-04-7005-437-6.

KIM, Inseong, SHIM, Joon Hyung y YANG, Jinkyu. Face Detection. En: Computer and Information Science. 2010. vol. 3, p. 71-90

KIM, Woo-Shik Kim; Dae-Sung Cho; Hyun Mun. Interplane prediction for RGB video coding. En: Image Processing, 2004. ICIP '04. 2004 International Conference on (Singapore, Singapore). Tertiary Interplane prediction for RGB video coding: IEEE, 2004

LABROSSE, Jean J. MicroC/OS-II: The Real Time Kernel. CRC Press, 2002. ISBN: 978-15-7820-103-7.

MAXFIELD, Clive. The Design Warrior's Guide to FPGAs: Devices, Tools and Flows. Elsevier, 2004. ISBN: 978-07-5067-604-5.

MOESLUND, Thomas B. Introduction to Video and Image Processing: Building Real Systems and Applications. Springer, 2012. ISBN: 978-14-4712-502-0.

MOSLEHPOUR, Saeid, JENAB, Kouroush y SILIVERI, E. H. Design and Implementation of NIOS II System for Audio Application. En: International Journal of Engineering and Technology (IACSIT). 2013. vol. 5, p. 627-634

NAVABI, Zainalabedin. Digital Design and Implementation With Field Programmable Devices. Springer, 2004. ISBN: 978-14-0208-011-1.

NELSON, Anthony Edward. Implementation of Image Processing Algorithms on FPGA Hardware. Nashville: (2000), p. 79.

ONVISION. ONDM72V2812L36HD User Guide. 2015

POYNTON, Charles. Digital Video and HD: Algorithms and Interfaces. Morgan Kaufmann, 2003. ISBN: 978-15-5860-792-7.

RASHID, Syed Sameer, DIXIT, Swati R. y DESHMUKH, A.Y. VHDL Based Canny Edge Detection Algorithm. En: International Journal of Current Engineering and Technology. 2014. vol. 4, no. 2, p. 749-752

ROMANO, David. Make: FPGAs: Turning Software Into Hardware With Eight Fun and Easy DIY Projects. Maker Media, Inc, 2016. ISBN: 978-14-5718-785-8.

SERFA JUAN, Ronnie O., PARK, Chan Su, KIM, Hi Seok y CHA, Hyeong Woo. FPGA Implementation for Real Time Sobel Edge Detector Block Using 3-Line Buffers. En: International Journal of Industrial Electronics and Electrical Engineering. dec, 2015. vol. 3, p. 84-89

SHIN, Min C., CHANG, Kyong I. y TSAP, Leonid V. Does colorspace transformation make any difference on skin detection? En: Proceedings of IEEE Workshop on Applications of Computer Vision (2002-Janua). Tertiary Does colorspace transformation make any difference on skin detection?: IEEE, 2002. p. 275-279

SOLOMON, Chris y BRECKON, Toby. Fundamentals of Digital Image Processing. Wiley, 2005. ISBN: 978-04-7084-472-4.

SZELISKI, Richard. Computer Vision: Algorithms and Applications. Springer, 2010. ISBN: 978-18-4882-934-3.

THAKUR, Sayantan, PAUL, Sayantanu, MONDAL, Ankur, DAS, Swagatam y ABRAHAM, Ajith. Face detection using skin tone segmentation. En: 2011 World Congress on Information and Communication Technologies. Tertiary Face detection using skin tone segmentation: IEEE, 2011. p. 53-60

VAN DEN BRANDEN LAMBRECHT, Christian J. Vision Models and Applications to Image and Video Processing. (2010).

VEZHNEVETS, Vladimir, SAZONOV, Vassili y ANDREEVA, Alla. A Survey on Pixel-Based Skin Color Detection Techniques. En: Proceedings of GraphiCon 2003. 2003. vol. 85, p. 85-92

YANG, Ming Hsuan, KRIEGMAN, David J. y AHUJA, Narendra. Detecting faces in images: A survey. En: IEEE Transactions on Pattern Analysis and Machine Intelligence. 2002. vol. 24, p. 34-58

ANEXO 1.
REVISIÓN BIBLIOGRÁFICA

CONCEPTOS DE IMAGEN

Dimensiones de una imagen.
Una característica importante de las imágenes y el video digital es que son señales multidimensionales. En el estudio del procesamiento de señales digitales, estas usualmente son funciones unidimensionales que dependen del tiempo. Sin embargo, las imágenes son funciones de dos e incluso tres dimensiones espaciales, mientras que el video digital consta de tres dimensiones espaciales y una de tiempo[61]. En conclusión, la dimensión de una imagen es el número de coordenadas que son requeridas para ubicar un punto dentro de esta, es por esto que el procesamiento de imágenes digitales y especialmente de video es tan complejo, requiriendo gran cantidad de recursos de hardware y software[62].

Tamaño de una imagen.
La cantidad de información que manejan las señales visuales generalmente es bastante grande e incrementa geométricamente con la dimensión de los datos. Esto afecta todos aspectos del procesamiento de imagen y video; siendo el volumen de datos el mayor problema en el procesamiento, almacenamiento, transmisión y visualización de la información.

El volumen de información para secuencias de video es bastante alto para que pueda ser procesado en tiempo real por microcontroladores, requiriendo de procesadores de alta velocidad o hardware dedicado. El prototipo a realizar en la tarjeta de desarrollo Altera DE2, aprovecha el componente de captura de video análogo para digitalizar la señal, y la FPGA para implementar los componentes personalizados para el procesamiento.

El tamaño de una matriz de pixeles de dos dimensiones junto con el tamaño de los datos almacenados para cada pixel de la imagen determina la resolución espacial y cuantización de color de la imagen[63]. El tamaño de una imagen depende de su resolución, la cual se puede expresar en términos de tres cantidades:

- Resolución espacial: El número de columnas por el número de filas de una imagen define el número de pixeles usados para cubrir el espacio visual capturado en la imagen.

---

[61] BOVIK. Op. cit, SOLOMON y BRECKON. Op. cit.
[62] BOVIK. Op. cit.
[63] Ibíd.

- Resolución temporal: Para un sistema de captura continuo como el video, este es el número de imágenes capturadas en un periodo de tiempo determinado.

- Resolución de bit: Esta define el número de posibles valores de intensidad o color que el pixel puede tener y se refiere a la cuantización de la información de la imagen. La resolución de bit comúnmente hace referencia al número de bits binarios necesarios para almacenar un nivel de cuantización dado[64].

Una imagen de escala de grises de dimensiones $\square \times \square$ (filas × columnas) y B bits de resolución, requiere de $\square \times \square \times \square$ bits para su almacenamiento. En cuanto al video, el sistema de video NTSC muestra la información a una velocidad de 23,976 *FPS* (cuadros por segundo), por motivos relacionados con la latencia de la visión humana, para frecuencias menores se puede apreciar un parpadeo. Una secuencia de video de $640 \times 480 \times 3 \times 24$ ocupa 21,6 MBytes por cada segundo de reproducción. Dos horas de una película en formato digital con la misma resolución, la cual es lejana a la calidad del cine, requeriría de 85 GBytes para su almacenamiento[65]. Si cada pixel fuera procesado de manera secuencial, se requeriría de 7.372.000 operaciones por segundo para lograr procesar la secuencia en tiempo real.

ESPACIOS DE COLOR

RGB.

Son arreglos de tres dimensiones que en concepto se pueden considerar como tres planos distintos de dos dimensiones, cada uno correspondiente a uno de los canales de colores rojo (R), verde (G) y azul (B). El espacio de color RGB es el que se usa más comúnmente para la representación de imágenes digitales, pues este convenientemente corresponde a los tres colores primarios, los cuales se mezclan para mostrar la imagen en un monitor u otro dispositivo similar.

El espacio de color RGB está basado en la parte del espectro magnético que es visible para los humanos. El espacio RGB es sustancialmente difícil de trabajar, esto debido a que no está relacionado con la forma natural en la cual percibimos los colores. Como una alternativa podemos usar un espacio de color que representa la percepción de color como es HSV, pues este separa las

---

[64] SOLOMON y BRECKON. Op. cit.
[65] BOVIK. Op. cit.

componentes de intensidad y color para aprovechar la mayor sensibilidad que posee la visión humana a los cambios de luz.[66].

La información de color RGB se puede convertir a escala de gris usando la siguiente fórmula[67]:

$$Y = 0{,}299R + 0{,}587G + 0{,}114B$$

Donde Y representa la componente luma de la imagen y los coeficientes que acompañan a cada componente R, G, y B respectivamente están dados por estándares para la transmisión de video, como por ejemplo ITU BT.601 y reflejan la sensibilidad del ojo humano a los diferentes colores. Por ejemplo, en la expresión dada, el coeficiente para el componente verde es mayor debido a que el ojo humano es más sensible para la luz verde en comparación a los componentes de azul y rojo.

HSV.
Los espacios de color perceptual son una alternativa para la representación de una imagen a color debido a que se asemejan más a la forma natural para la percepción humana y la comprensión del color que el espacio RGB. Existen varias alternativas para la representación del color, concretamente el espacio de color HSV es muy popular para el análisis de imágenes.

Cada uno de los tres parámetros se pueden interpretar de la siguiente manera:

- H (matiz o color) es la longitud de onda de color dominante, por ejemplo, rojo, verde, azul.

- S (saturación) es la “pureza” del color (en el sentido de la cantidad de luz blanca que se mezcla con esta).

- V (valor) es la intensidad del color (también conocida como luminancia).

CMY (Cian, Magenta, Amarillo).
El modelo CYM está basado en los colores secundarios y es un esquema de color de sustracción, pues los colores C, M y Y se restan del blanco puro para obtener el color requerido. Por esta razón, es utilizado frecuentemente como un modelo de color en las impresoras donde el blanco es el punto de inicio (0, 0, 0)[68].

---

[66] SOLOMON y BRECKON. Op. cit.
[67] DAWSON-HOWE. Op. cit.
[68] Ibíd.

YUV y YCbCr.

El espacio de color YUV se utiliza en la transmisión de las señales de televisión análoga (PAL, NTSC…) y está compuesto de la luma (Y) junto con dos componentes de color: azul menos luma (U) y rojo menos luma (V). La transformación desde el espacio RGB se realiza de la siguiente forma[69]:

$$\begin{bmatrix} \square \\ \square \\ \square \end{bmatrix} = \begin{bmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ -0{,}147 & -0{,}298 & 0{,}436 \\ 0{,}615 & -0{,}515 & -0{,}1 \end{bmatrix} \cdot \begin{bmatrix} \square \\ \square \\ \square \end{bmatrix} \quad \begin{matrix} \square \in [0{,}255] \\ \square \in [-111{,}111] \\ \square \in [-157{,}157] \end{matrix}$$

$$\begin{bmatrix} \square \\ \square \\ \square \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1{,}14 \\ 1 & -0{,}395 & -0{,}581 \\ 1 & 2{,}032 & 0 \end{bmatrix} \cdot \begin{bmatrix} \square \\ \square \\ \square \end{bmatrix} \quad \begin{matrix} \square \in [0{,}255] \\ \square \in [0{,}255] \\ \square \in [0{,}255] \end{matrix}$$

El espacio de color YCbCr se utiliza para la transmisión, almacenamiento y compresión de imagen y video, como por ejemplo los estándares JPEG y MPEG respectivamente. La conversión del espacio de color RGB se realiza como se muestra a continuación:

$$\begin{bmatrix} \square \\ \square_{\square} \\ \square_{\square} \end{bmatrix} = \begin{bmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ -0{,}169 & -0{,}331 & 0{,}5 \\ 0{,}5 & -0{,}419 & -0{,}081 \end{bmatrix} \cdot \begin{bmatrix} \square \\ \square \\ \square \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} \quad \begin{matrix} \square \in [0{,}255] \\ \square_{\square} \in [0{,}255] \\ \square_{\square} \in [0{,}255] \end{matrix}$$

$$\begin{bmatrix} \square \\ \square \\ \square \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1{,}403 \\ 1 & -0{,}344 & -0{,}714 \\ 1 & 1{,}773 & 0 \end{bmatrix} \cdot \begin{bmatrix} \square \\ \square_{\square} - 128 \\ \square_{\square} - 128 \end{bmatrix} \quad \begin{matrix} \square \in [0{,}255] \\ \square \in [0{,}255] \\ \square \in [0{,}255] \end{matrix}$$

El valor 128 se suma o resta con el fin de que los valores estén dentro del rango [0,255][70].

IMÁGENES BINARIAS

Morfología de las imágenes binarias.

a. Operaciones lógicas: Los operadores morfológicos se pueden interpretar como operaciones lógicas dentro de un grupo local de pixeles. Los operadores lógicos se aplican en la operación de un elemento estructurante (el cual hace referencia a una figura con una forma característica) usado para probar o interactuar con una imagen dada[71].

---

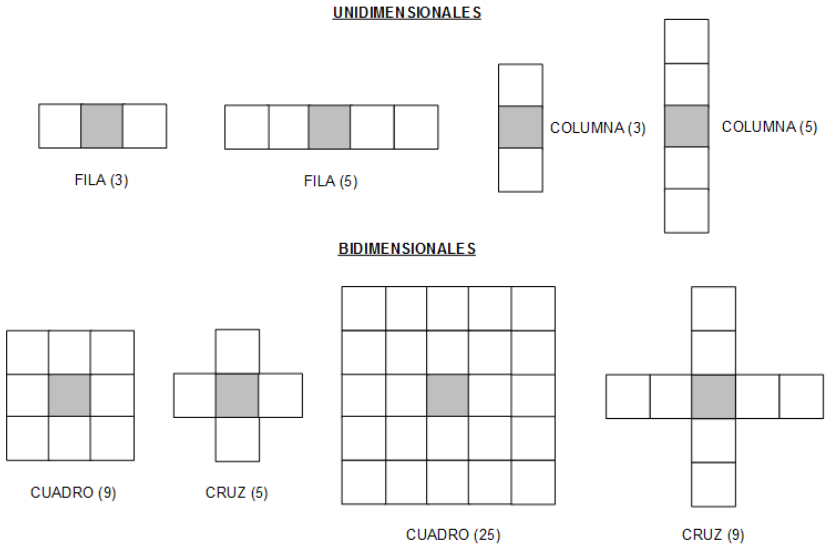
[69] Ibíd, MOESLUND. Op. cit.
[70] MOESLUND. Op. cit.
[71] BOVIK. Op. cit.

b. Elemento estructurante: Las operaciones morfológicas cambian la forma de los objetos utilizando operaciones morfológicas locales. Debido a que existen operadores locales, una metodología formal se debe definir para garantizar que las operaciones ocurran dentro del espacio deseado. El mecanismo para garantizar esta condición se conoce como elemento estructurante.

   Un elemento estructurante define una regla geométrica mediante la cual se obtiene información de la vecindad de un pixel dado. También se conoce como ventana, pues a menudo se visualiza como un grupo de pixeles vacíos en movimiento, que pasa sobre la imagen.

   En la Figura 47 se muestran algunos ejemplos de elementos estructurantes más utilizados.

Figura 47. Elementos estructurantes unidimensionales y bidimensionales



Fuente: BOVIK, Alan Conrad. Handbook of Image and Video Processing. Academic Press (2005).

Dada una imagen binaria □ y un elemento estructurante □, la ventana en la imagen de coordenadas □ está dada por:

$$□□(□) = \{□(□ - □); □ \in □\}$$

El cuál es el conjunto de pixeles de la imagen cubiertos por □ cuando está centrado en las coordenadas □.[72]

[72] Ibíd.

En la Tabla 16 se muestran ejemplos de ventanas para algunos elementos estructurantes ilustrados en la Figura 47.

Tabla 16. Ejemplos elemento estructurante y ventanas.

Elemento Estructurante	Ventana
$\mathbb{B} = \text{FILA}(3)$	$\mathbb{BB}(\mathbb{B}_1, \mathbb{B}_2) = \{\mathbb{B}(\mathbb{B}_1, \mathbb{B}_2 - 1), \mathbb{B}(\mathbb{B}_1, \mathbb{B}_2), \mathbb{B}(\mathbb{B}_1, \mathbb{B}_2 + 1)\}$
$\mathbb{B} = \text{COLUMNA}(3)$	$\mathbb{BB}(\mathbb{B}_1, \mathbb{B}_2) = \{\mathbb{B}(\mathbb{B}_1 - 1, \mathbb{B}_2), \mathbb{B}(\mathbb{B}_1, \mathbb{B}_2), \mathbb{B}(\mathbb{B}_1 + 1, \mathbb{B}_2)\}$
$\mathbb{B} = \text{CRUZ}(5)$	$\mathbb{BB}(\mathbb{B}_1, \mathbb{B}_2) = \{\mathbb{B}(\mathbb{B}_1 - 1, \mathbb{B}_2), \mathbb{B}(\mathbb{B}_1, \mathbb{B}_2 - 1), \mathbb{B}(\mathbb{B}_1, \mathbb{B}_2), \mathbb{B}(\mathbb{B}_1, \mathbb{B}_2 + 1), \mathbb{B}(\mathbb{B}_1 + 1, \mathbb{B}_2)\}$

Fuente: BOVIK, Alan Conrad. Handbook of Image and Video Processing. Academic Press (2005).

c. Operaciones morfológicas: Los operadores morfológicos que se usan con mayor frecuencia se describen a continuación:

- Dilatación: Es una técnica utilizada para expandir el número de pixeles de un objeto, generalmente en todas las direcciones simultáneamente. La operación de dilatación se representa como $\mathbb{B} \oplus \mathbb{B}$, donde $\mathbb{B}$ representa a la imagen binaria y $\mathbb{B}$ es el elemento estructurante.

- Erosión: Es una técnica utilizada para reducir el contorno del objeto removiendo pixeles del borde del mismo. La erosión se representa como $\mathbb{B} \ominus \mathbb{B}$, donde $\mathbb{B}$ representa a la imagen binaria y $\mathbb{B}$ es el elemento estructurante.

- Apertura: Es una operación de erosión seguida de la operación de dilatación con el mismo elemento estructurante: $\mathbb{B} \circ \mathbb{B} = (\mathbb{B} \ominus \mathbb{B}) \oplus \mathbb{B}$. A diferencia de la dilatación y erosión, está mantiene aproximadamente el tamaño del objeto.

- Cierre: Es la operación de dilatación seguida de una operación de erosión, manteniendo el elemento estructurante: $\mathbb{B} \cdot \mathbb{B} = (\mathbb{B} \oplus \mathbb{B}) \ominus \mathbb{B}$.[73].

d. Filtros: Los filtros morfológicos son filtros Booleanos. Dada una imagen $\mathbb{B}$, una función Booleana $\mathbb{B}$ y un elemento $\mathbb{B}$, la imagen Booleana filtrada $\mathbb{B} = \mathbb{B}(\mathbb{B})$ está dada por:

$$\mathbb{B}(\mathbb{B}) = \mathbb{B}[\mathbb{BB}(\mathbb{B})]$$

[73]DAWSON-HOWE. Op. cit.

Donde □ se encuentra sobre el dominio de la imagen. Así, para cada □ el filtro recolecta los pixeles locales de acuerdo a la geometría del elemento estructurante, realizando una operación Booleana en ellos y devolviendo el resultado Booleano □(□).

Las operaciones Booleanas (AND, OR, las más comunes), se utilizan para crear filtros morfológicos los cuales actúan sobre los objetos de la imagen para su conformación expandiendo o contrayéndolos, suavizándolos y eliminando los detalles pequeños[74].

e. Detección de bordes: Los filtros son bastante efectivos para suavizar las imágenes binarias, pero estos tienen otras aplicaciones importantes como la detección de bordes.

   Inicialmente la detección de bordes puede parecer trivial, pues los bordes pueden ser definidos como la transmisión de 1 a 0 y viceversa. Sin embargo, con la presencia de ruido la detección de bordes se dificulta bastante, pues es bastante sensible a este fenómeno[75].

***Tresholding* (método del valor umbral)**

Una imagen binaria se obtiene generalmente de una imagen en escala de grises mediante procesos de extracción de información. Si el proceso se realiza correctamente se puede mejorar significativamente el procesamiento, análisis o interpretación de la imagen.

El proceso más simple de extracción se conoce como *tresholding* y se puede pensar como una forma de cuantización extrema. Suponiendo que una imagen □ en escala de grises puede tomar □ posibles niveles de gris $\{0,1,2,\dots,□-1\}$. Luego se define un valor de umbral □, que se encuentre dentro del rango de la escala de grises $□ \in \{0,1,2,\dots,□-1\}$. El *thresholding* crea una imagen binaria □(□,□) comparando la intensidad □(□,□) de cada pixel de la imagen □ de acuerdo al siguiente criterio:

$$□(□,□) = \begin{cases} 1 \text{ □□ } □(□,□) > □ \\ 0 \text{ □□ □□□□ □□□□} \end{cases}$$

Una pregunta muy frecuente es como encontrar un valor apropiado de □ para realizar un exitoso *thresholding* y como saber si el resultado que se obtiene es el adecuado. Está pregunta tiene una respuesta relativa, sin embargo, en todos los

---

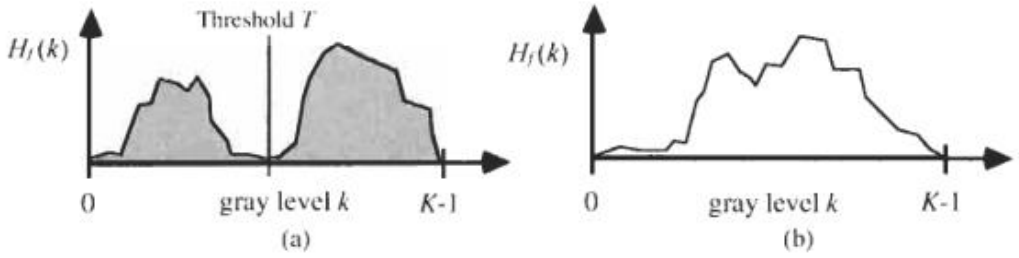
[74] BOVIK. Op. cit.
[75] Ibíd.

casos la herramienta básica para entender el proceso de *thresholding* es el histograma de la imagen.

El *thresholding* se utiliza con más frecuencia y es más efectivo en imágenes que tienen un histograma bimodal. Estos comúnmente se asocian a imágenes que contienen objetos y fondos con una gran diferencia de luminosidad entre ellos. Un ejemplo de histograma bimodal se puede apreciar en la ilustración ubicada en la parte izquierda de la Figura 48

Para una imagen con un histograma bimodal, una estrategia para realizar un *thresholding* adecuado es ubicar el umbral *T* entre los modos de la imagen, como se muestra en la Figura 48.

Figura 48. a) Histograma con modos claramente separados (histograma bimodal) y b) Histograma con modos levemente separados

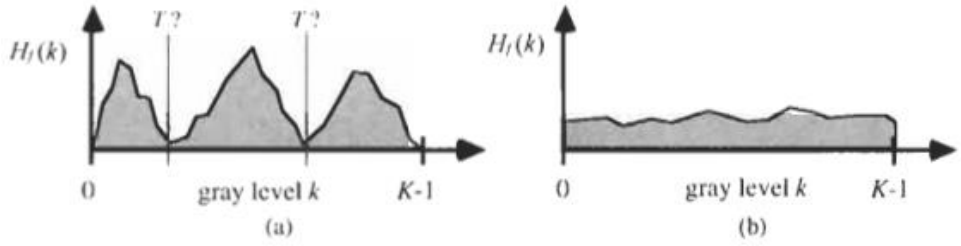


Fuente: BOVIK, Alan Conrad. Handbook of Image and Video Processing. Academic Press (2005).

El problema se vuelve más complejo si la imagen contiene múltiples modos o si la imagen es no modal como se representa en la Figura 49. Estas imágenes no son susceptibles a simples procesos de *Thresholding*, especialmente si lo que se quiere conseguir es una separación del fondo de la imagen del resto de ella[76].

[76] Ibíd, SOLOMON y BRECKON. Op. cit.

Figura 49. a) Histograma multimodal y b) Histograma no modal



Fuente: BOVIK, Alan Conrad. Handbook of Image and Video Processing. Academic Press (2005).

VIDEO ANÁLOGO Y DIGITAL

Muestreo de video.

El procesamiento digital de video requiere que el flujo de este sea muestreado y cuantizado. El muestreo de video requiere tomar información a través de una dimensión nueva y diferente, el tiempo. Por tanto, necesitamos de nuevas técnicas y conceptos.

La dimensión del tiempo tiene una dirección diferente a las dimensiones de espacio, las cuales normalmente se consideran sin dirección. Frecuentemente el video es procesado en tiempo real, lo que significa que los resultados del proceso aparecen instantáneamente. Este procesamiento no puede depender de demasiadas muestras futuras de video, sin embargo, este puede procesar la información lo suficientemente rápido como para que el resultado parezca instantáneo.

Inicialmente se tiene una señal de video análogo $□(□,□,□)$ donde $(□,□)$ son las coordenadas de espacio y $□$ es el tiempo en ambas dimensiones continuas, debido a que el flujo de radiación que incide en el dispositivo de captura de video es continuo en escenarios normales de observación. Sin embargo, el video que se muestra en un monitor realmente no es análogo, pues es muestreado en una dimensión de espacio y la dimensión de tiempo.

Prácticamente los llamados sistemas de video análogo, como la televisión y monitores, representan el video como una señal $□(□)$ de una dimensión. Antes de ser mostrada, la señal unidimensional se obtiene de muestrear $□(□,□,□)$ a través del espacio vertical y el tiempo. Esto se conoce como escaneo y el resultado es una serie de muestras de tiempo, las cuales son imágenes completas o cuadros (*frames*), donde cada uno está compuesto de muestras espaciales o líneas de escaneo (*scanlines*).

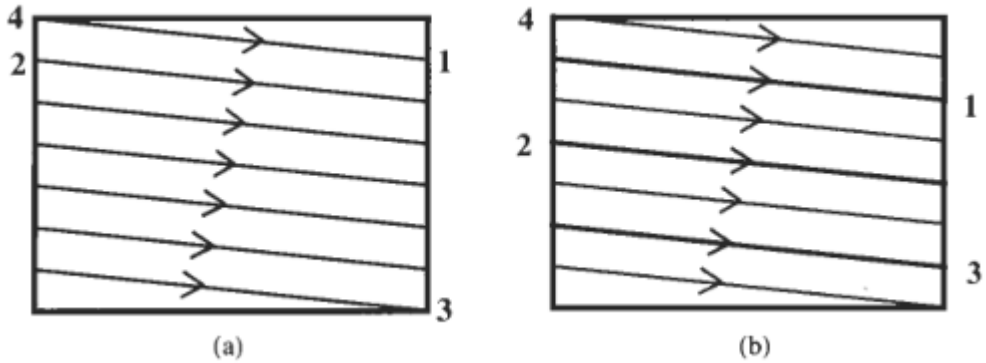
Los tipos de escaneo de video más usados son: escaneo progresivo y escaneo entrelazado. El escaneo progresivo traza un cuadro completo, línea por línea

desde arriba hacia abajo a una tasa de escaneo de Δ⍰ s/cuadro, como por ejemplo los monitores de computador de alta resolución.

Para ambos tipos de escaneo, la frecuencia de actualización es la velocidad de cuadros por segundo a la cual la información se muestra en un monitor o pantalla. Es importante que la velocidad de cuadros por segundo sea lo suficientemente alta, de lo contrario al ver el video este podría presentar un fenómeno de parpadeo o intermitencia.

El escaneo entrelazado presenta una solución a los sistemas que no pueden alcanzar una gran frecuencia de cuadros por segundo. En el entrelazado P:1, toda línea P se actualiza en cada cuadro de refresco. Los subcuadros en el video entrelazado se conocen como campos; por tanto, P campos conforman un cuadro. El entrelazado más común es el 2:1, el cual se usa en los sistemas de televisión estándar. Por tanto, si la velocidad del cuadro es de 30 Hz, entonces la velocidad del campo será de 60Hz.

Figura 50. Escaneo de video progresivo contra escaneo de video entrelazado.



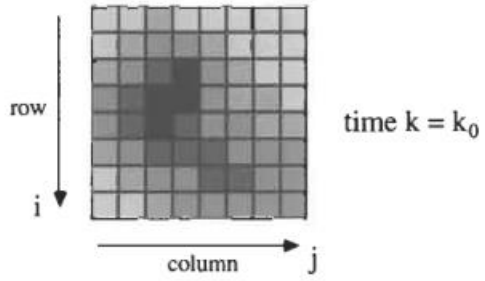
Fuente: BOVIK, Alan Conrad. Handbook of Image and Video Processing. Academic Press (2005).

La Figura 50 muestra gráficamente el proceso para el escaneo progresivo y entrelazado. Para el primer caso, al terminar la línea 1 salta inmediatamente a la línea 2; después de terminar la línea 3 salta directamente a 4 y mediante un pulso de sincronización, las señales inician el escaneo de otro cuadro. Para el escaneo entrelazado, los líneas rojas y azules se escanean de forma alterna, al finalizar el escaneo de 1 este va inmediatamente a 2; al terminar el campo azul en 3 se dirige a 4 para iniciar el escaneo de un nuevo campo[77].

[77] BOVIK. Op. cit.

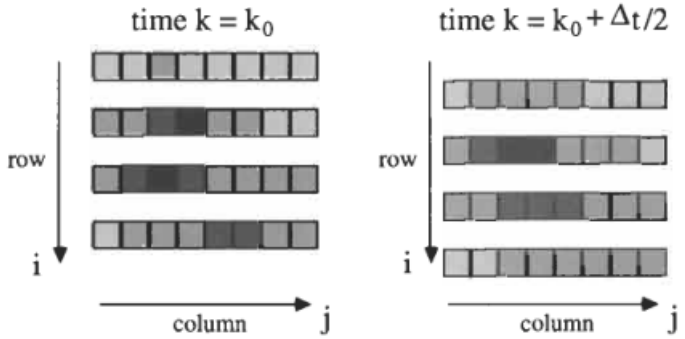
Si un video análogo es muestreado, entonces el muestreo se organiza de forma rectangular e indexada, de una manera intuitiva como se muestra en la Figura 51. Si un video análogo es muestreado de forma entrelazada, entonces el video digital se entrelaza como en la Figura 52.

Figura 51. Cuadro de una secuencia de video muestreada de forma progresiva



Fuente: BOVIK, Alan Conrad. Handbook of Image and Video Processing. Academic Press (2005).

Figura 52. Cuadro de una secuencia de video muestreada de forma entrelazada

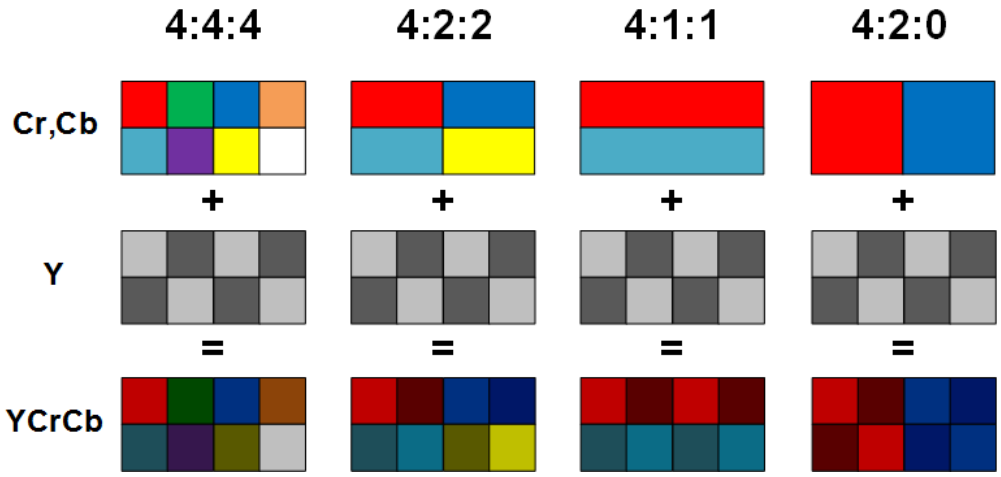


Fuente: BOVIK, Alan Conrad. Handbook of Image and Video Processing. Academic Press (2005).

Submuestreo de crominancia.
La agudeza cromática de la visión humana es significativamente más baja que la agudeza acromática. Para aprovechar esta característica, los colores primarios rara vez son usados para codificar el color directamente. En cambio, se calculan las señales de diferencia de color (croma), de esta forma es posible aplicar diferentes muestreos espaciales a los componentes de color y luminancia de la

imagen para lograr un menor tamaño[78]. La notación y características de los tipos de muestreo más usados comúnmente son:

Figura 53. Ejemplos de submuestreo de la componente de color



Fuente: POYNTON, Charles. Digital Video and HD: Algorithms and Interfaces. Morgan Kaufmann (2003).

- 4:4:4. La imagen muestra en su primera columna que para cada una de las componentes posee la misma taza de muestreo, lo cual hace referencia al espacio de color RGB, pues para este no aplica el submuestreo de las componentes.

- 4:2:2. El espacio de color YCbCr utiliza el muestreo 4:2:2, donde las componentes Cb y Cr se submuestrean horizontalmente a la mitad de la taza de muestreo de la componente Y.

- 4:1:1. Ciertos sistemas de video utilizan este tipo de muestreo, donde las componentes Cb y Cr son submuestreadas horizontalmente a un cuarto de la taza de muestreo de la componente Y.

- 4:2:0. Las componentes Cb y Cr son submuestreadas horizontal y verticalmente a la mitad de la taza de muestreo de la componente Y. No

[78] VAN DEN BRANDEN LAMBRECHT, Christian J. Vision Models and Applications to Image and Video Processing. (feb., 2010).

existe diferencia en cuanto a la taza datos en comparación con el submuestreo 4:1:1.[79].

El decodificador de video ADV7180 incluido en la tarjeta de desarrollo Altera DE2 entrega los datos en el formato 4:2:2. Posteriormente, y para el desarrollo del prototipo se utiliza el submuestreo 4:4:4, simplificando el procesamiento de cada canal como se explica en 2.2.2.

Sistemas NTSC y PAL.
Para el tratamiento de video se podría utilizar las componentes del espacio de color RGB, pero estas resultan demasiado robustas para almacenar, procesar o transmitir la información pues sus componentes de color no se submuestrean. Por lo anterior, las componentes luma y croma son usadas para esta tarea, permitiendo submuestrear las componentes de color y dejar intacta la componente luma y así hacer un uso eficiente de los recursos disponibles. Incluso después de realizar el submuestreo de las componentes de color, la señal de video presenta un gran flujo de información.

NTSC (National Television System Committee) es el sistema de televisión analógico utilizado en Norteamérica, Centroamérica, parte de Suramérica y Japón. PAL (Phase Alternating Line) es el sistema de codificación utilizado para la transmisión de televisión analógica para la mayor parte del mundo, incluyendo a Europa Occidental.

La codificación de las componentes de video para los sistemas NTSC y PAL se creó para resolver tres necesidades principales: primero limitar el ancho de banda para la transmisión de video; segundo permitir la recepción de la transmisión de señales a color minimizando las pérdidas de información; y finalmente para adaptar los receptores a color para recibir las transmisiones a blanco y negro.[80].

Codificación de video análogo.
Para la codificación de los sistemas NTSC o PAL se debe tener en cuenta los siguientes pasos, los cuales representan a grandes rasgos el proceso:

- Las componentes de la señal son ordenadas y filtradas para formar las señales de luma (Y) y diferencia de color (U y V, por ejemplo).

- U y V son moduladas sobre un par de ondas continuas subportadoras de color profundamente relacionadas para producir una señal croma modulada.

---

[79] POYNTON. Op. cit.
[80] Ibíd.

- Las componentes de luma y croma modulada son sumadas para formar la señal NTSC o PAL. La suma de luma y croma es susceptible a contener cierto grado de interferencia mutua[81].

Decodificación de video análogo.
La decodificación de los sistemas NTSC y PAL involucra de manera general, los siguientes pasos:

- Las señales de luma y croma modulada son separadas usando filtros. Alternativamente, se puede utilizar una frecuencia intercalada para mejorar la separación.

- La señal croma es demodulada para producir las componentes de diferencia de color U y V.

- Si se requiere las componentes RGB, las componentes de diferencia de color son interpoladas, luego la luma y las componentes de diferencia de color son extraídas de su matriz[82].

PROCESAMIENTO DE VIDEO

Segmentación de video.
De manera similar a la segmentación de imágenes, la segmentación de video hace referencia a la identificación de regiones en un cuadro de video que es homogéneo. Diferentes características y criterios de homogeneidad conducen a diferentes segmentaciones de la misma información. No hay garantía de que una segmentación sea útil, pues una región puede tener múltiples colores, texturas o movimiento. La segmentación de movimiento es un método de etiquetado de pixeles en el que cada cuadro está asociado con un movimiento independiente de una parte de una escena.

La segmentación de movimiento está fuertemente relacionada con dos problemas: la detección de movimiento y la estimación de movimiento. La detección de movimiento es un caso especial de la segmentación de movimiento con solo dos regiones, llamadas regiones con cambio o sin cambio (para el caso de cámaras estáticas) o regiones de movimiento local y global (en el caso de cámaras en movimiento). La detección y segmentación de movimiento poseen las mismas limitaciones asociadas a la estimación de movimiento: oclusión y apertura. Por ejemplo, los pixeles en una región pueden parecer estáticos incluso si están en movimiento como resultado de un problema de apertura; o clasificaciones

[81] Ibíd.
[82] Ibíd.

erróneas pueden ser asignadas a pixeles de una región de una imagen como resultado de un problema de oclusión[83].

Seguimiento de objetos.
El seguimiento de uno o varios objetos en una secuencia de video es un problema importante en la visión computacional y ha sido abordado de diferentes formas, algunos de estos enfoques se describen a continuación:

- Búsqueda exhaustiva: Donde una imagen se sigue cuadro a cuadro, buscando el mejor resultado en cada uno de ellos.

- Promedio de movimiento: Donde se utiliza el histograma de la imagen para la comparación.

- Flujo óptico: Es una técnica utilizada para determinar la dirección en la cual todos los pixeles se mueven entre un cuadro y otro. Se utiliza frecuentemente para la predicción de movimiento.

- Seguimiento de características: Limitando las características del objeto a seguir solo aquellas zonas que son de interés. Esto puede proporcionar un enfoque veloz y más robusto[84].

SISTEMAS EMBEBIDOS

Sistemas embebidos SoPC (System on a Programmable Chip).
Los sistemas embebidos SoPC (*System on a Programmable Chip*) son un caso particular de sistema embebido, pues mezclan las ventajas entre la programación de hardware y software. Estos sistemas poseen un procesador de propósito general junto a dispositivos de entrada/salida, capaces de realizar tareas comunes y una interfaz de usuario general junto con aceleradores de hardware para procesar operaciones computacionalmente intensivas.

Cuando tanto el procesador como el hardware de aceleración se integran en un único circuito integrado, se le llama SoC (*System on a Chip*). El crecimiento de las capacidades computacionales ha hecho posible el diseño y programabilidad de los dispositivos de hardware. En el caso de que el hardware de aceleración sea programable, se le conoce como SoPC (*System on a Programmable Chip*).

El diseño de sistemas basados en procesadores de propósito general convencionales requiere la selección de un procesador, periféricos y aceleradores

---

[83] BOVIK. Op. cit.
[84] DAWSON-HOWE. Op. cit.

de hardware específicos para la construcción de la plataforma de hardware. Debido a la selección limitada de dispositivos disponibles, la arquitectura fija del procesador y el costo de manufacturar circuitos impresos, la configuración del hardware es rígida y las funcionalidades especificas suelen implementarse en software personalizado.

El desarrollo de las FPGA (*Field Programmable Gate Array*) cambió el paradigma de desarrollo para los sistemas embebidos. El hardware puede ser modificado y adecuado a las necesidades, el procesador puede ser personalizado, seleccionando únicamente los componentes necesarios y la interfaz de E/S puede ser adecuada al diseño. El hardware se personaliza a través de un lenguaje de descripción de hardware o HDL (*Hardware Description Language*), el cual se sintetiza y se aplica a la FPGA. De esta manera el diseño de sistemas embebidos basado en SoPC con FPGA, provee una nueva dimensión de flexibilidad ya que tanto el software como el hardware pueden personalizarse para necesidades particulares[85].

Existen dos amplias categorías de dispositivos FPGA: reprogramables y programables una vez o *one-time programmable* (OTP). Dependiendo de la tecnología usada para configurar e ingresar el programa diseñado dentro de la FPGA existen 4 tecnologías diferentes detalladas en la Tabla 17.

Tabla 17. Tecnologías de programación de FPGA

Tecnología de programación	**Descripción y características**
Basado en SRAM	Un dispositivo externo (una memoria no volátil o un microprocesador) programa el dispositivo durante el encendido. Permite una reconfiguración rápida. La configuración es volátil. Algunos dispositivos pueden reconfigurarse mientras están encendidos.
Basado en antifusibles	Se configuran “quemando” fusibles internos para implementar la funcionalidad deseada. La configuración es no volátil y no puede ser modificada.
Basada en EPROM	Similar a los dispositivos EPROM. La configuración es no volátil. El dispositivo debe configurarse por fuera del circuito.

[85] CHU, Pong P. Embedded SoPC Design With Nios II Processor and VHDL Examples. Wiley, 2011. ISBN: 978-11-1800-888-1.

Tecnología de programación	Descripción y características
Basada en EEPROM	Similar a los dispositivos EEPROM. La configuración es no volátil. El dispositivo puede configurarse y reconfigurarse por fuera del circuito.

Fuente: COFER, R. C.; HARDING, Benjamin F. Rapid System Prototyping With FPGAs. Newnes (2005).

Para los proyectos de diseño y desarrollo de prototipos, las soluciones de FPGA basadas en SRAM son la mejor elección por la velocidad de lectura y escritura de la memoria además de su facilidad para programación por la interfaz de la FPGA con la esta. Aunque las FPGA de tipo OTP tienen aplicaciones en ambientes específicos, no son adecuadas debido a los cambios y modificaciones que el diseño puede tener durante su concepción e implementación. Por lo anterior es recomendable que la FPGA seleccionada sea reprogramable[86].

Arquitectura de las FPGA.
Los dispositivos FPGA se basan en un número de estructuras configurables comunes. Aunque cada fabricante ofrece variaciones sobre la implementación de estas estructuras, estas son comunes a casi todos los dispositivos FPGA del mercado. Las partes fundamentales son las siguientes:

- Bloques o celdas lógicas.
- Matriz de enrutamiento y señales globales.
- Bloques de E/S.
- Recursos de reloj.
- Multiplicadores.
- Memoria.
- Características avanzadas.

[86] COFER, R. C. y Benjamin F. HARDING. Rapid System Prototyping With FPGAs. Newnes, 2005. ISBN: 978-07-5067-866-7.

DISEÑO DE SISTEMAS EMBEBIDOS

El diseño de hardware en FPGA es un proceso necesariamente iterativo e implica realizar un balance entre los requisitos del sistema a implementar y los 3 principales limitantes que son: velocidad a la cual necesita trabajar o procesar la información el sistema a implementar; consumo energético del sistema para desarrollar la tarea para la cual fue diseñado; y área destinada para el dispositivo final[87]. Los diferentes algoritmos deben ajustarse a las características técnicas de la FPGA, puesto que, al momento de ser sintetizados, puede suceder que se usen más recursos que los disponibles.

La creciente complejidad de los sistemas embebidos ha obligado a las herramientas de diseño a ofrecer mayores niveles de abstracción[88]. Para el caso de Altera, la herramienta Quartus II incluye la posibilidad de realizar sistemas embebidos a través de la aplicación Qsys. Esta permite el diseño de sistemas embebidos en un alto nivel de abstracción y facilita la creación e integración de componentes personalizados[89].

Métodos de diseño.
Existen diversas formas de abordar el diseño de sistemas embebidos, en general, estos métodos se pueden reducir a dos tipos principales: el diseño de “arriba hacia abajo” (*top-down*) o de “abajo hacia arriba” (*bottom-up*).

El diseño *top-down,* también conocido como diseño jerárquico, en él se comienza desde el más alto nivel de abstracción. Este nivel superior puede representarse a través de un diagrama de bloques, como HDL o una mezcla de estos. Cada bloque o unidad de diseño puede a su vez contener sub-bloques y funciones lógicas que los conecten. De esta manera, un diseño complejo se puede dividir en partes más simples de implementar.

En el diseño *bottom-up* se comienza con una sección del diseño, se realizan todas las tareas y detalles necesarios para su funcionamiento y luego se continúa el trabajo en otra sección. También se conoce como “diseño plano”. Se suelen usar diagramas esquemáticos para la representación del diseño. Pueden existir varios diagramas esquemáticos conectados entre sí, pero todos están a un alto nivel de detalle. Este método funciona bien para diseños pequeños, pero para el caso de

---

[87] KILTS, Steve. Advanced FPGA Design: Architecture, Implementation, and Optimization. Wiley-IEEE Press, 2007. ISBN: 978-04-7005-437-6.
[88] Ibíd.
[89] ALTERA CORPORATION. Quartus II Handbook Version 13.1 Volume 1: Design and Synthesis. San Jose, CA: Altera Corporation (nov, 2013).

sistemas embebidos, en donde componentes de diferentes desarrolladores interactúan entre sí, el método *top-down* es el adecuado[90].

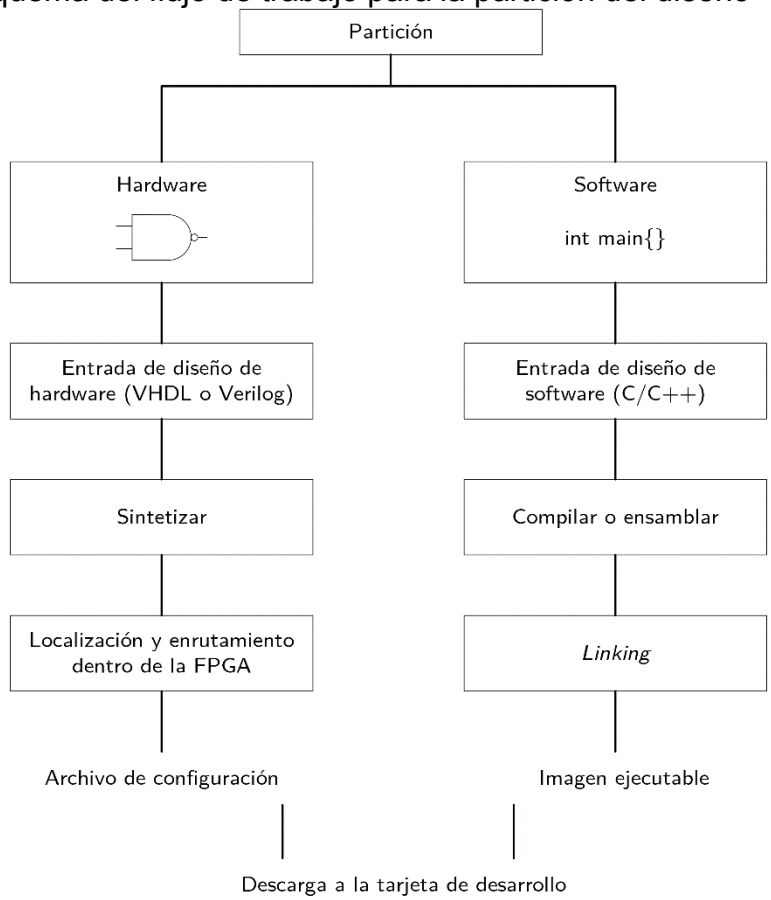
En el desarrollo del prototipo se utiliza el método más adecuado dependiendo de los requerimientos de cada componente. El diseño de cada uno de ellos se realiza mediante *bottom-up*, para posteriormente ensamblar el sistema final mediante el diseño *top-down* gracias a la herramienta Qsys incluida en el paquete Quartus II™.

Partición del diseño en hardware/software.
El primer paso para el diseño de sistemas embebidos es la partición, es decir, decidir que partes del sistema se implementan en hardware o en software.

La Figura 54, muestra un flujo de trabajo básico para los diseños de software o de hardware, así como los diferentes pasos que implica cada uno.

[90] ROMANO, David. Make: FPGAs: Turning Software Into Hardware With Eight Fun and Easy DIY Projects. Maker Media, Inc, 2016. ISBN: 978-14-5718-785-8.

Figura 54. Esquema del flujo de trabajo para la partición del diseño



Fuente: MAXFIELD, Clive. The Design Warrior's Guide to FPGAs: Devices, Tools and Flows. Elsevier (2004).

Cualquier parte de un diseño electrónico puede desarrollarse en hardware, usando elementos lógicos combinatorios y secuenciales, o software, usando instrucciones que se ejecutan en un microprocesador. A continuación, se listan algunos criterios a tener en cuenta para la selección del tipo de implementación, hardware o software, en el diseño[91]:

- Complejidad: Algoritmos sofisticados que requieren recursión o que requieren un número de iteración indefinido antes de su ejecución, son más apropiados para una implementación en software.

- Velocidad: Ejecutar un algoritmo en software siempre acarrea un *overhead* o consumo adicional de recursos sin relación al algoritmo principal. Los

[91] MAXFIELD. Op. cit.

algoritmos implementados en hardware, por el contrario, pueden ser optimizados para una baja latencia o alto ancho de banda.

- Repetición: Una tarea que se repita muchas veces, aunque no sea compleja pueden influir sobre la velocidad del procesador, por tanto, este se puede beneficiar delegando estas tareas a un hardware dedicado.

- Precisión en tiempo real: Un microprocesador ejecuta instrucciones en un orden especifico y toda tarea debe esperar a ser ejecutada. Si una tarea requiere una precisión temporal que se mida en ciclos de reloj, una implementación de hardware es la única que garantiza tal nivel de precisión.

- Sistema operativo o interfaz de usuario: En caso de requerirse un sistema operativo, se hace necesario un microprocesador ya sea físico o sintetizado[92].

Entrada y captura del diseño.
Los métodos más usados para la entrada del diseño son la captura esquemática y los lenguajes de descripción de hardware o *hardware description languages* (HDL). La síntesis de estos elementos permite que puedan ser implementados dentro de la FPGA independientemente del fabricante. Por esto, los lenguajes de descripción de hardware (HDL) proveen mejor reúso del diseño; facilidad en el bosquejo y creación de prototipos de sistemas complejos, gestión unificada de los elementos del sistema de hardware, elementos que incrementan la productividad en comparación al diseño de hardware que no utiliza elementos programables.

Los HDL pueden usarse tanto para la implementación como para la simulación. Es importante tener en cuenta que ciertas estructuras y características de HDL no pueden ser traducidas al hardware físico. Como consecuencia, un diseño compilable no necesariamente es sintetizable, es decir que no necesariamente se puede trasladar al hardware provisto por el dispositivo, por esto es necesario conocer las limitaciones del lenguaje y los elementos que pueden implementarse en hardware[93].

HDL provee facilidad para el diseño de sistemas complejos, pues se pueden gestionar y revisar fácilmente; además

---

[92] KILTS. Op. cit.
[93] COFER y HARDING. Op. cit.

Núcleos de propiedad intelectual (IP).
La propiedad intelectual en el contexto de las FPGA se puede definir como un bloque de diseño reutilizable con un rango fijo de funcionalidad. El término *core of intellectual property* (CIP) usualmente se refiere a bloques de diseño funcionales verificados, obtenidos de un grupo por fuera del equipo de diseño.

Entre las fuentes de IP se encuentran los fabricantes de FPGAs, distribuidores independientes y recursos de código abierto. La IP se puede distribuir en diferentes formatos bajo un amplio rango de acuerdos de licencia, también incluyen documentación y soporte dependiendo de la fuente. Existen tres tipos comúnmente reconocidos de IP: *soft*, *firm* y *hard*. Las definiciones de estos se presentan en la Tabla 18[94].

Tabla 18. Tipos de IP

Tipo	Descripción
Núcleo *soft* (*soft core*)	La funcionalidad del diseño esta implementada directamente HDL, con pocas o ninguna optimización específica para un fabricante o dispositivo en particular
Núcleo firme (*firm core*)	Diseño implementado con un HDL optimizado a un dispositivo o clase de dispositivos especifica. La optimización puede tener incluso un enfoque específico (rendimiento, área o consumo de energía)
Núcleo *hard* (*hard core*)	Implementada en lógica fija en vez de usar los bloques de la FPGA, la funcionalidad se implementa durante la fabricación del dispositivo, por esta razón no puede ser modificada o removida.

Fuente: COFER, R. C.; HARDING, Benjamin F. Rapid System Prototyping With FPGAs. Newnes (2005).

Según su función la IP se puede agrupar dentro de tres amplias categorías. Estas categorías se diferencian por la aplicación o funcionalidad: procesamiento, especialización e interfaz.

Entre los núcleos IP se encuentran procesadores tradicionales de 8, 16 y 32 bits, procesadores especializados diseñados específicamente para aplicaciones de procesamiento digital de señales o *digital signal processing* (DSP), componentes

[94] Ibíd.

orientados al procesamiento siguiendo el modelo de ruta de datos (*data path*). Las interfaces de E/S pueden ser seriales o paralelas, permitiendo elegir la velocidad del protocolo necesario para estándares que van desde UART hasta Ethernet[95].

El rango de disponibilidad de IP se puede dividir en amplias categorías que se superponen entre sí, la Tabla 19 muestra algunas de estas junto con ejemplos.

Tabla 19. Categorías de IP

Categoría	**Ejemplos**
Funciones de procesamiento digital de señales (DSP)	Decodificador Viterbi, FFT, MAC, FIR, DCT
Funciones matemáticas	CORDIC, multiplicador paralelo, divisor segmentado
Funciones básicas	Registro de corrimiento, acumulador, comparador, sumador
Funciones de memoria	Módulo de bloque de memoria, módulo de memoria distribuido
Procesamiento de imagen	Conversión de espacio de color, codificador de JPEG
Comunicaciones	Cifrado AES, codificador Reed-Solomon, decodificador Turbo
Procesadores	RISC, CISC, compatible con 8051, compatible con Z80
Dispositivos	UART, VGA, temporizador, PCI, USB, I2C, CAN

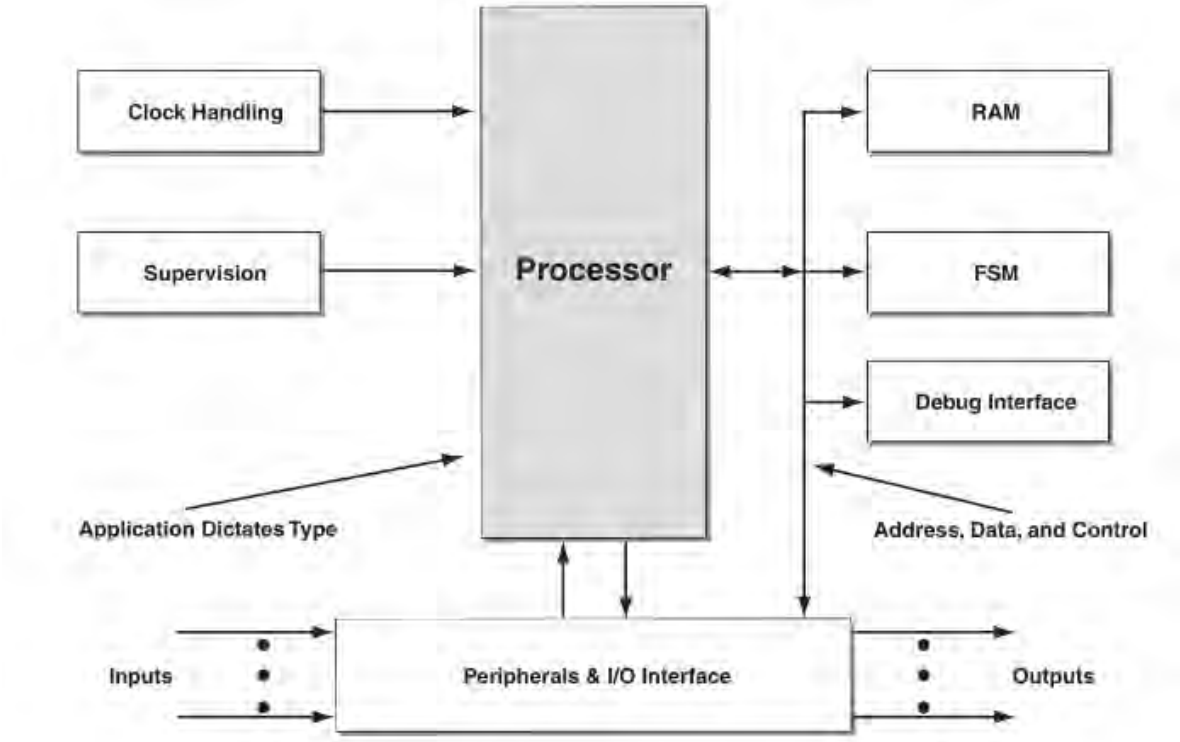
Fuente: COFER, R. C.; HARDING, Benjamin F. Rapid System Prototyping With FPGAs. Newnes (2005).

Procesadores embebidos.
Los procesadores son uno de los componentes más flexibles y útiles para un diseñador de sistemas embebidos. La flexibilidad del diseño de procesadores ha evolucionado a través de la estandarización de hardware y software y los avances de la tecnología. La arquitectura RISC o computador con conjunto de instrucciones reducidas es probablemente una de las más implementadas en la actualidad. Junto con la arquitectura RISC, herramientas de software robustas y lenguajes de programación de alto nivel han permitido el uso de procesadores en casi cualquier tipo de sistema embebido[96].

[95] Ibíd.
[96] Ibíd.

Entre las ventajas de incluir un procesador dentro de la FPGA se encuentra una menor obsolescencia, menor cantidad de componentes en el diseño final y un mayor control sobre las características finales del dispositivo. La Figura 55 muestra una implementación potencial de componentes dentro de una FPGA.

Figura 55. Potencial implementación de un sistema dentro de una FPGA

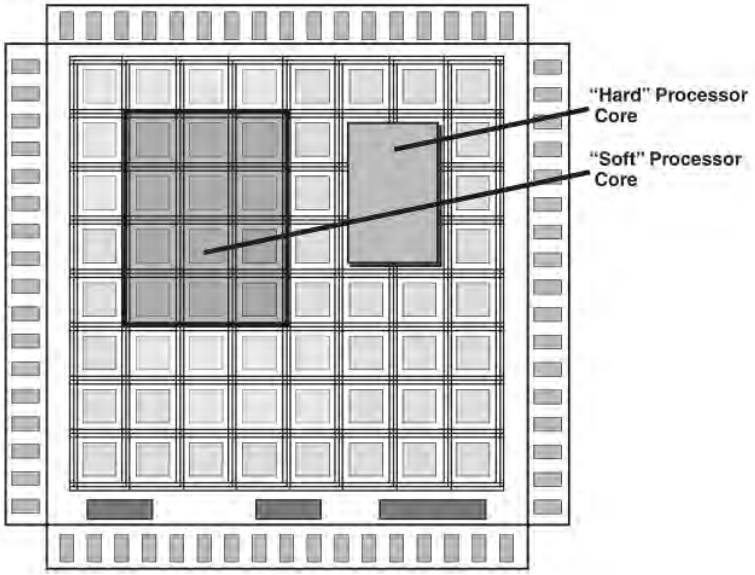


Fuente: COFER, R. C.; HARDING, Benjamin F. Rapid System Prototyping With FPGAs. Newnes (2005).

La flexibilidad de las FPGA reprogramables se puede mejorar aún más incluyendo procesadores dentro de la misma. Este procesador embebido se puede implementar como un núcleo *soft* o *hard*. Los núcleos *soft* son implementaciones de procesadores realizadas en HDL sin ninguna optimización específica a la arquitectura de destino; estos tienen por lo general un menor rendimiento y son menos eficientes en términos de uso de recursos. Los núcleos *hard* también son implementaciones en HDL, pero han sido optimizadas para una arquitectura específica. Los procesadores Nios II de Altera y MicroBlaze de Xilinx son ejemplos de núcleos de procesador *hard*. Por último, los núcleos *hard* son un IP diseñada al nivel de compuertas lógicas con función fija dentro de la FPGA. La Figura 56, muestra de forma esquemática las diferencias entre los procesadores *hard* y *soft*,

la implementación del procesador *hard* se hace aparte de la lógica de la FPGA, y se comunica a través de interconexiones definidas, mientras que en los procesadores *soft*, los diferentes componentes del procesador se encuentran implementados dentro de los bloques de construcción de la FPGA. [97].

Figura 56. Diferencia entre un procesador *hard* y *soft*



Fuente: COFER, R. C.; HARDING, Benjamin F. Rapid System Prototyping With FPGAs. Newnes (2005).

DESARROLLO CON LA TARJETA ALTERA DE2

DISEÑO CON QUARTUS II

Altera ofrece la suite Quartus II para el desarrollo en sus FPGAs. Esta aplicación abarca todos los pasos necesarios para implementar un diseño, desde la captura y entrada de la descripción del hardware, simulación y programación del dispositivo[98].

Definición del proyecto.
Dentro de la aplicación Quartus II cada diseño se define como un proyecto. Este abarca los archivos de diseño organizados de manera jerárquica y encabezados por un archivo de diseño de nivel superior (*top-level design*), este archivo describe la interfaz del dispositivo de hardware, sus entradas, salidas y arquitectura.

---

[97] Ibíd.
[98] NAVABI, Zainalabedin. Digital Design and Implementation With Field Programmable Devices. Springer, 2004. ISBN: 978-14-0208-011-1.

Un proyecto también define el dispositivo de destino, con el fin de que el programa tenga conocimiento de las características, limitaciones y optimizaciones necesarias para lograr sintetizar e implementar el diseño[99].

Entrada del diseño.
Quartus II ofrece diferentes formas para introducir el diseño, entrada a través de un lenguaje de descripción de hardware (HDL – *Hardware Description Language*), entrada esquemática, uso de partes existentes, uso de Megafunciones o *IP Cores*, o una mezcla de estos.

El software Quartus II incluye editores de texto y esquemáticos, así como plantillas HDL. Soporta archivos de diseño VHDL, Verilog, SystemVerilog y archivos esquemáticos de bloques. Todos los elementos anteriores se pueden combinar entre sí, con archivos de IP de terceros, incluyendo la combinación de componentes dentro de un sistema Qsys[100].

Análisis y verificación.
La verificación funcional del hardware se realiza a través de un paquete de simulación, este por lo general es independiente de la herramienta de desarrollo. Sin embargo, Altera incluye una versión básica del paquete MentorGraphics el cual simula el comportamiento y respuesta de un componente de hardware descrito en HDL a los estímulos descritos dentro de un *testbench*, presentando los resultados en un diagrama de onda digital[101].

Compilación.
Después de que un diseño es ingresado con éxito y sus resultados de simulación pre-síntesis han sido verificados, es necesario compilarlo para acercarlo a la implementación en hardware.

El proceso de compilación traduce las partes del diseño descritas en diferentes métodos a un formato intermedio (fase de análisis). Une todas las partes, genera la lógica correspondiente (fase de síntesis), ubica y enruta las partes dentro del dispositivo de destino y genera los detalles de temporización[102].

Diseño de sistemas embebidos usando Qsys.
Qsys es una herramienta de integración incluida en el paquete Quartus II™. Qsys captura diseños de hardware para sistemas embebidos a un nivel de abstracción

---

[99] Ibíd.
[100] ALTERA CORPORATION, Quartus II Handbook Version 13.1 Volume 1: Design and Synthesis. Op. cit.
[101] NAVABI. Op. cit.
[102] Ibíd.

alto y automatiza la tarea de definir e integrar diferentes componentes HDL personalizados. Estos componentes pueden ser núcleos IP (*IP cores*), Megafunciones y otros módulos de diseño. Qsys facilita el reúso de los diseños empaquetando e integrando los componentes personalizados con los componentes de Altera y componentes IP de terceros.

La herramienta se encarga de crear automáticamente la lógica de interconexión en base al tipo de conectividad de alto nivel que se especifique, eliminado así la necesidad de escribir HDL para especificar conexiones del sistema. Para tomar ventaja de Qsys, se diseñan los componentes utilizando interfaces estándar soportadas como el bus Avalon de Altera.

Cada componente puede tener cualquier número de interfaz. Dentro de Qsys, cada interfaz representa un conjunto de señales que se pueden conectar a otro componente dentro del sistema de Qsys o exportarse fuera del mismo. Los tipos de interfaz soportados se muestran en la Tabla 20.

Tabla 20. Tipos de interfaz Avalon

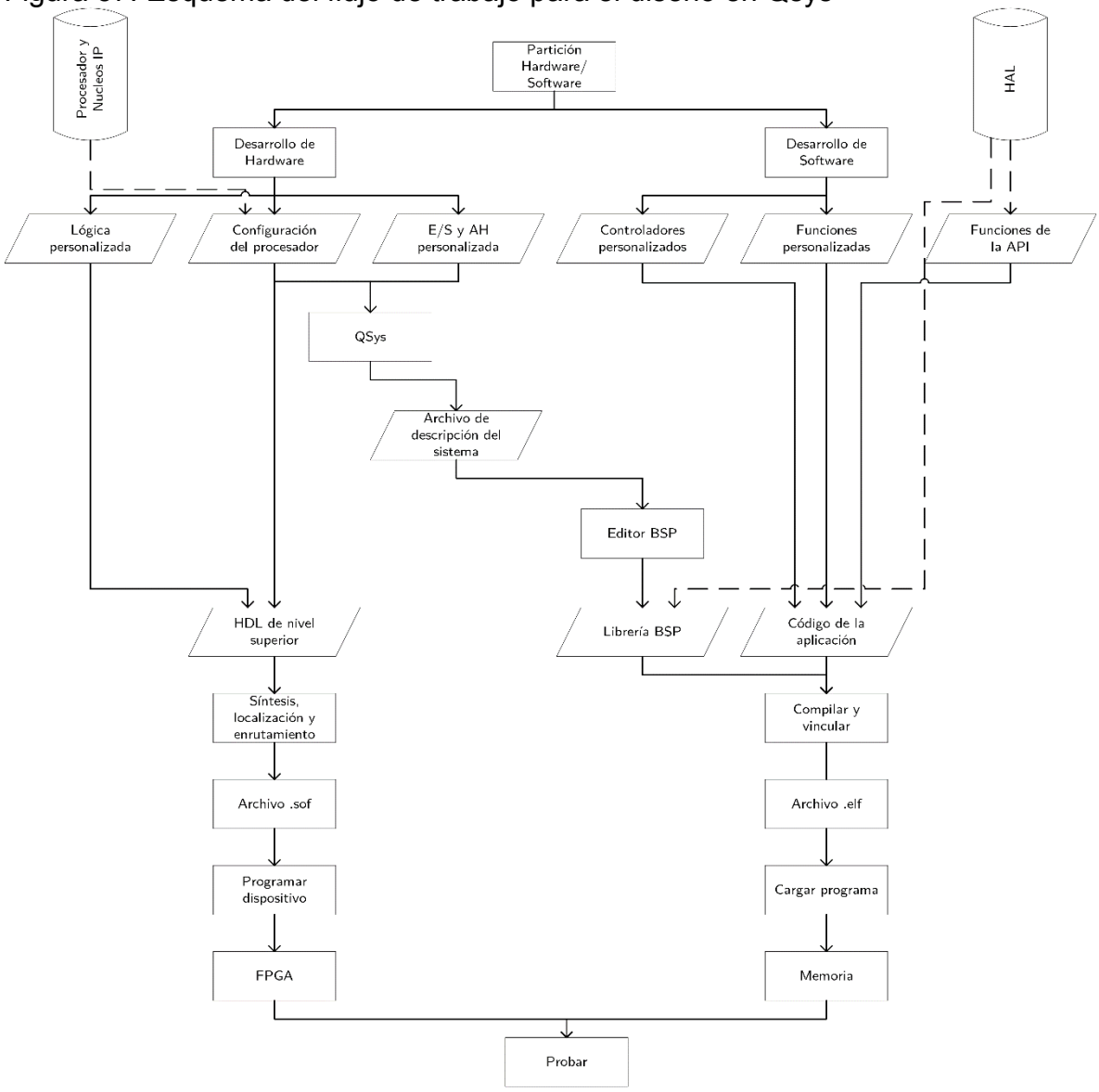
Tipo de interfaz	**Descripción**
Memory-Mapped **(Mapeada en memoria)**	Implementa una interconexión que provee vías concurrentes entre maestros y esclavos.
Streaming **(Flujo de datos)**	Conecta fuentes *source* y sumideros *sink* de la interfaz de flujo de *Avalon Streaming* (Avalon-ST), que transmiten datos de manera unidireccional, incluyendo flujos multicanal, paquetes y datos de DSP, permite definir ancho de bus, paquetes y condiciones de error.
Interrupts **(Interrupciones)**	Conecta generadores y receptores de interrupciones. Qsys soporta peticiones de interrupción de un bit (IRQs).
Clocks **(Relojes)**	Conecta interfaces de salida de reloj con interfaces de entrada.
Resets	Conecta fuentes de *reset* con entradas de *reset*, Qsys inserta la lógica adecuada dependiendo si la señal es síncrona o asíncrona.
Conduits **(Conductos)**	Los conductos representan señales que se exportan del sistema Qsys, se usan por lo general para señales de E/S que no son parte de una interfaz estándar. Las interfaces de conducto se muestran en el nivel superior del sistema con el fin de conectarse a los componentes de E/S y se pueden conectar a

Tipo de interfaz	Descripción
	dispositivos externos o lógica de la FPGA definida por fuera del sistema Qsys.

Fuente: ALTERA CORPORATION. Avalon Interface Specifications. San Jose, CA: Altera Corporation (2015).

La Figura 57 muestra un flujo de trabajo típico para un diseño Qsys con componentes personalizados, en este flujo, los componentes se crean y se depuran antes de ser integrados, siguiendo un esquema de diseño de abajo hacia arriba (*bottom-up*).

Figura 57. Esquema del flujo de trabajo para el diseño en Qsys



Fuente: CHU, Pong P. Embedded SoPC Design With Nios II Processor and VHDL Examples. Wiley (2011).

PROCESADOR NIOS II

Nios II es el procesador propietario de Altera creado específicamente para sus dispositivos FPGA. Es configurable y puede ser reducido para suplir necesidades particulares. A diferencia de un procesador fijo prefabricado, Nios II es un procesador de núcleo *soft*; es decir, esta descrito con lenguaje HDL y transferido a las celdas lógicas de la FPGA, del tipo de los descritos en la sección 0. Este enfoque ofrece mayor flexibilidad, pues un procesador de núcleo *soft* se puede

configurar agregando o quitando características dependiendo del sistema, con el propósito de alcanzar las metas de rendimiento o costos del diseño[103].

El procesador Nios II sigue los principios de la arquitectura RISC (*Reduced Instruction Set Computer*) y usa un número reducido de instrucciones. Sus principales características son[104]:

- Arquitectura de carga-almacenamiento.
- Formato de instrucción fijo de 32 bits.
- *Data path* interno de 32 bits.
- Espacio de direcciones de 32 bits.
- Espacio de E/S mapeado en memoria.
- Solicitud de interrupción de 32 niveles.
- 32 registros de propósito general.

Versiones del procesador Nios II.
Altera permite utilizar 3 versiones del procesador Nios II, una comparativa de sus características se muestra en la Tabla 21 y se resumen a continuación[105]:

- Nios II/f: La versión rápida (*fast*) del núcleo está diseñada para un rendimiento óptimo. Tiene un *pipeline* de 6 niveles, caché de instrucciones y datos, y predicción de saltos dinámica (*dynamic branch prediction*)
- Nios II/s: La versión estándar balancea el tamaño con el rendimiento. Tiene un *pipeline* de 5 niveles, caché de instrucciones, y predicción de saltos estática (*static branch prediction*).

---

[103] MOSLEHPOUR, Saeid, et al. Design and Implementation of NIOS II System for Audio Application. En: International Journal of Engineering and Technology (IACSIT). 2013. vol. 5, CHU, Embedded SoPC Design With Nios II Processor and VHDL Examples. Op. cit.
[104] CHU, Embedded SoPC Design With Nios II Processor and VHDL Examples. Op. cit, ALTERA CORPORATION. Nios II Classic Processor Reference Guide. San Jose, CA: Altera Corporation (jun, 2015).
[105] ALTERA CORPORATION, Nios II Classic Processor Reference Guide. Op. cit.

- Nios II/e: El núcleo “económico” (*economy*) está diseñado para un uso mínimo de espacio dentro de la FPGA, no tiene ni *pipeline* ni caché.

Tabla 21. Comparación de versiones de Nios II

	Nios II/e	**Nios II/s**	**Nios II/f**
Pipeline	1 nivel	5 niveles	6 niveles
Predicción de saltos	-	Estática	Dinámica
Multiplicación	Software	Multiplicador de 3 ciclos	Multiplicador de 1 ciclo
Corrimiento	Software	Corrimiento tipo barril de 3 ciclos	Corrimiento tipo barril de 1 ciclo
Caché de instrucciones	-	0.5 KB a 64 KB	0.5 KB a 64 KB
Caché de datos	-	-	0.5 KB a 64 KB

Fuente: CHU, Pong P. Embedded SoPC Design With Nios II Processor and VHDL Examples. Wiley (2011).

Cada versión puede ser personalizada a un nivel más detallado, incluyendo o quitando ciertas características (como unidad de depuración JTAG) y ajustando el tamaño y rendimiento de ciertos componentes, como tamaño de caché. Las 3 versiones a pesar de sus diferencias comparten el mismo conjunto de instrucciones. De esta forma las versiones se comportan de igual forma y el software no necesita modificarse para una versión en particular[106].

BUS AVALON

La plataforma Altera SoPC utiliza la estructura de interconexión Avalon para conectar el procesador y varios módulos IP. Para un periférico de E/S personalizado o hardware de aceleración, es posible agregar un circuito *wrapper* compatible con la especificación Avalon y hacer que sea compatible con Qsys. De esta manera el componente se puede utilizar como un núcleo IP normal y ser integrado a un sistema Nios II.

La plataforma Qsys de Altera define un conjunto de interfaces estandarizadas conocidas como interfaces Avalon, estas se realizan por la lógica de interconexión

[106] CHU, Embedded SoPC Design With Nios II Processor and VHDL Examples. Op. cit.

usando las características de la FPGA, esta lógica es realizada automáticamente por el software Qsys y elimina los problemas causados por un bus centralizado[107].

Tipos de buses Avalon.
El estándar Avalon consiste en las siguientes interfaces[108]:

- *Avalon memory mapped interface* (Avalon MM): Define una conexión maestro-esclavo basada en dirección de memoria. Un maestro Avalon MM usa una dirección para identificar a un esclavo Avalon MM y puede leer o escribir datos al esclavo.

- *Avalon streaming interface* (Avalon ST): Define un enlace unidireccional entre dos componentes. Una fuente (*source*) Avalon ST transmite datos a un sumidero (*sink*) Avalon ST de forma continua.

- *Avalon memory mapped tri-state interface*: Se puede considerar un esclavo Avalon MM especial usado para controlar buses y periféricos con lógica tri-estado (*tri-state*)

- *Avalon clock*: Define las señales de reloj (*clock*) y reinicio (*reset*) utilizadas por un componente. Una interfaz de salida de reloj Avalon genera la señal de reloj y una interfaz de entrada de reloj Avalon la recibe.

- *Avalon interrupt*: Esta interfaz permite que los esclavos envíen eventos a los maestros. Un transmisor de interrupciones genera la solicitud y un receptor las acepta.

- *Avalon conduit*: Agrupa y exporta señales al exterior del sistema Qsys.

DESARROLLO DE SOFTWARE CON NIOS II EDS

La inclusión de un procesador Nios II en un sistema embebido implica el desarrollo de software para el sistema, para este fin Altera incluye la plataforma Nios II EDS (*Embedded Design Suite*) que incluye la suite de herramientas GNU adaptada para el procesador Nios II. La plataforma consiste en:

- Compilador basado en GCC con las utilidades binarias GNU.

---

[107] Ibíd.
[108] ALTERA CORPORATION. Avalon Interface Specifications. San Jose, CA: Altera Corporation (dec, 2015).

- Una versión de las librerías *newlib* de C adaptadas para procesadores Nios II.

- Una interfaz para controladores simple conocida como HAL (*Hardware Abstraction Layer* – Capa de Abstracción de Hardware).

- El sistema operativo MicroC/OS II adaptado para el procesador Nios II.

Las herramientas de compilación de software (SBT – *Software Build Tools*) incluyen estas aplicaciones, ya sea en línea de comandos, en interfaz gráfica o a través del entorno de desarrollo integrado (IDE – *Integrated Design Enviroment*) SBT basado en la IDE Eclipse. Esta simplifica en gran medida los procesos de desarrollo, compilación y depuración de las soluciones de software para el procesador Nios II.

Un proyecto de software Nios II contiene dos partes principales: las aplicaciones del usuario y un paquete de soporte de tarjeta (BSP – *Board Support Package*). Las aplicaciones consisten en el código realizado por el diseñador para el sistema mientras que el BSP son las librerías y código específico para la implementación del procesador dentro del proyecto. El BSP es generado por las herramientas de desarrollo a partir del archivo de descripción del sistema generado con la herramienta Qsys[109].

El módulo JTAG incluido con el procesador Nios II junto con el cable USB Blaster incluido con la tarjeta de desarrollo permite al software Nios II EDS un método único y consistente de comunicación con el procesador. De esta forma no es necesario crear mecanismos de interfaz para depurar o diagnosticar el procesador embebido[110].

Nios II SBT para Eclipse.
Nios II SBT para Eclipse es una capa GUI que ejecuta las utilidades y scripts de Nios II SBT dentro de un ambiente de desarrollo unificado, basado en la herramienta de desarrollo de código abierto Eclipse. Esto permite lograr todas las tareas relacionadas con el desarrollo de software como crear, editar, compilar, ejecutar y depurar sin la necesidad de aplicaciones externas.

Los programas desarrollados con el lenguaje C/C++ consisten de un proyecto de aplicación, proyectos de librerías de usuarios personalizados y un proyecto BSP.

---

[109] CHU, Embedded SoPC Design With Nios II Processor and VHDL Examples. Op. cit.
[110] ALTERA CORPORATION, Nios II Classic Software Developer's Handbook. Op. cit.

El resultado final del desarrollo es un archivo *Executable and Linking Format* (.elf) el cual se ejecuta dentro del procesador Nios II[111].

Es posible utilizar la memoria Flash Externa para almacenar la aplicación compilada y de esta forma asegurar que el programa se mantenga aun cuando se interrumpa el suministro de energía. Para este fin Nios II SBT provee la utilidad de programación flash. Para esto se requiere que el dispositivo flash se encuentre correctamente configurado dentro del sistema Qsys e indicar al procesador que inicie la ejecución desde el dispositivo flash. De esta forma el *bootloader* programado carga el programa a la memoria del sistema y permite que la aplicación se ejecute sin la necesidad de ser descargada nuevamente a través del USB Blaster.

BOARD SUPORT PACKAGE (BSP)

Un proyecto Nios II BSP es una librería especializada que contiene código de apoyo específico del sistema. Un BSP provee un ambiente de ejecución personalizado para un procesador en un sistema de hardware Nios II. Las EDS provee las herramientas necesarias para modificar la configuración que controla el comportamiento de la BSP[112].

Un BSP contiene los siguientes elementos:

- Capa de abstracción de hardware (HAL).
- Librería *newlib* de C opcional personalizada.
- Controladores de dispositivo.
- Paquetes de software opcionales.
- Sistema operativo en tiempo real opcional.

CAPA DE ABSTRACCIÓN DE HARDWARE (HAL)

La capa de abstracción de hardware es un entorno de ejecución embebido que prosee una interfaz de controlador sencilla que permite a los programas conectarse con el hardware en el que se ejecutan. La interfaz de programación de aplicación (API – *Application Programming Interface*) está integrada con la librería

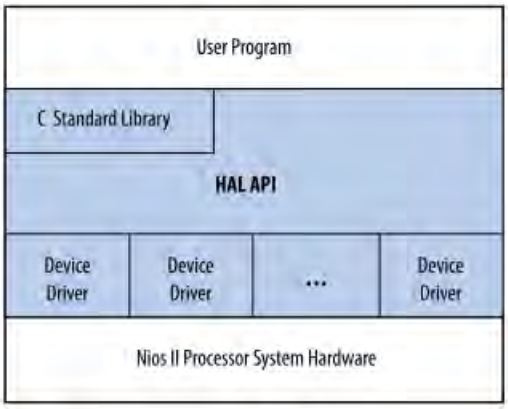
---

[111] Ibíd.
[112] Ibíd.

estándar ANSI C, la API permite el acceso a dispositivos utilizando funciones de C familiares como `printf()`, `fopen()`, `fwrite()`, etc.

La HAL presenta una interfaz coherente que separa los detalles de bajo nivel de los programas de aplicación como se muestra en la Figura 58 y permite abstraer operaciones como escritura de registros e inicialización de componentes.

Figura 58. Capa de un sistema basado en HAL



Fuente: ALTERA CORPORATION. Nios II Classic Software Developer's Handbook. San Jose, CA: Altera Corporation (2015).

Los servicios provistos por la HAL son[113]:

- Integración con la librería ANSI C a través de *newlib*.
- Controladores de dispositivos.
- API HAL: Provee una interfaz consistente a servicios como acceso a dispositivos y manejo de interrupciones.
- Inicialización del sistema: Provee tareas de inicialización para el procesador y entorno de ejecución antes de la ejecución del punto de inicio de la aplicación `main()`.
- Inicialización de dispositivos: Instancia e inicializa los dispositivos del sistema antes de la ejecución de `main()`.

[113] Ibíd.

SISTEMA OPERATIVO EN TIEMPO REAL

Para la mayoría de los sistemas embebidos, un sistema operativo de una sola tarea resulta demasiado restrictivo. Lo que se requiere es un sistema operativo que pueda ejecutar múltiples aplicaciones simultáneamente y proveer control y comunicación entre las tareas.

Los sistemas operativos en tiempo real disponen de características adicionales que les permiten a las aplicaciones que normalmente interactúan con el procesador de forma directa, hacerlo sin que el sistema operativo interfiera con sus actividades. La principal característica de los sistemas en tiempo real es que tienen un tiempo de respuesta estrictamente definido. Si un dispositivo genera una interrupción, el sistema la atenderá en un tiempo definido y este tiempo no es afectado por la carga de trabajo del sistema operativo[114].

MicroC/OS II.
MicroC/OS II es un *kernel* en tiempo real producido por Micrium Inc. Es un *kernel* portable, escalable, apropiativo (*pre-emptive*), multitarea y en tiempo real[115]. Es usado en una gran cantidad de aplicaciones comerciales y esta implementado en más de 40 arquitecturas diferentes incluida el procesador Nios II[116].

MicroC/OS II provee las siguientes características:

- Tareas (hilos).
- Banderas de eventos.
- Intercambio de mensajes.
- Administración de memoria.
- Semáforos.
- Administración de temporización del procesador.

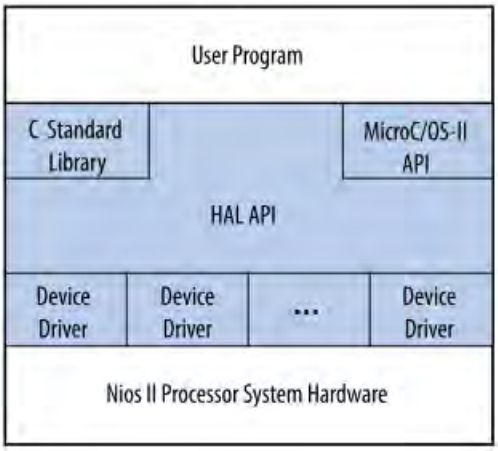
---

[114] HEATH, Steve. Embedded Systems Design. Newnes, 2003.
[115] LABROSSE, Jean J. MicroC/OS-II: The Real Time Kernel. CRC Press, 2002. ISBN: 978-15-7820-103-7, ALTERA CORPORATION, Nios II Classic Software Developer's Handbook. Op. cit.
[116] ALTERA CORPORATION, Nios II Classic Software Developer's Handbook. Op. cit.

El *kernel* MicroC/OS II opera por encima de la capa de abstracción de hardware del BSP del procesador Nios como se muestra en la Figura 59. Debido a esto el desarrollo en MicroC/OS II para el procesador Nios II tiene las siguientes ventajas:

- Las aplicaciones son portables a otros sistemas de hardware Nios II.
- Los programas son resistentes a cambios del hardware sobre el que se encuentran.
- Los programas pueden acceder a todos los servicios provistos por la HAL, permitiendo usar su API

Figura 59. Arquitectura de programas realizados sobre el sistema operativo MicroC/OS II



Fuente: ALTERA CORPORATION. Nios II Classic Software Developer's Handbook. San Jose, CA: Altera Corporation (2015).

El sistema operativo ofrece una alta relación de costo/beneficio y ha sido certificado para aplicaciones médicas y militares, este se provee en forma de código C con optimizaciones específicas del procesador en el que se implementa. Altera provee una versión optimizada específicamente para el procesador Nios II y se incluye con las herramientas Nios II EDS[117].

[117] KAMAL, Raj. Embedded Systems: Architecture, Programming and Design. New Delhi: Tata McGraw-Hill, 2009.