COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

ACELERACIÓN EN VHDL PARA FPGA DE ALGORITMOS DE DETECCIÓN DE BORDES (SOBEL / CANNY)

Autor: Martina Fraga Agras

Director: Fermín Zabalegui Sanz

Madrid

Junio de 2026

Declaración de originalidad

Declaro bajo mi responsabilidad que el Proyecto presentado con el título **Aceleración en VHDL para FPGA de algoritmos de detección de bordes (Sobel/Canny)** e la ETS de Ingeniería – ICAI de la Universidad Pontificia Comillas en el curso académico **4º** es de mi autoría y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Uso de Inteligencia Artificial[1]

Declaro bajo mi responsabilidad que (indicar la opción correcta):

☐ No he utilizado Inteligencia Artificial en la elaboración del presente documento.

- He utilizado Inteligencia Artificial en la elaboración del presente documento y/o del Anexo B siempre en las condiciones permitidas por la Universidad Pontificia Comillas, es decir, aplicando el Nivel 2 de la Escala de Evaluación de Perkins et al. (2024): "*La IA puede utilizarse para actividades previas a la tarea, como la lluvia de ideas, la descripción y la investigación inicial. Este nivel se centra en el uso de la IA para la planificación, las síntesis y la generación de ideas, pero las evaluaciones deben hacer hincapié en la capacidad de desarrollar y refinar estas ideas de forma independiente*". En concreto, las Inteligencia Artificial ha sido empleada para:

> -La ayuda a la comprensión de los algoritmos y los sistemas con Claude AI con el modelo Claude Sonnet 4.6, con Gemini Pro y con Chat GPT basado en GPT-5.5
> -La obtención de fuentes de información para la consulta de dudas y aporte de conocimiento con ChatGPT basado en GPT-5.5.
> -La resolución de ciertas dudas en el proceso de desarrollo del código de VHDL con Claude AI con el modelo Claude Sonnet 4.6.
> -Las comparaciones entre el algoritmo en Python y el desarrollo en VHDL se hicieron mediante scripts de Python hechos con Claude Code con el modelo Opus 4

[1] Esta declaración se refiere al uso de la Inteligencia Artificial generativa para realizar los documentos del Proyecto (Anexo B y Memoria). No aplica a Proyectos donde, por su naturaleza, deban emplear inteligencia artificial como parte de los mismos (aplicación de técnicas de aprendizaje automático, redes neuronales, análisis de datos…)


Firmado: Martina Fraga Agras
Fecha: 18/06/2026

Autorización para la entrega del Proyecto

El Director del Proyecto	El co-Director del Proyecto (si aplica)
ZABALEGUI SANZ FERMIN - 50202045Z 2026.06.18 18:49:37 +02'00'	
Fdo: Fermín Zabalegi Sanz	Fdo:
Fecha: 18/06/26	Fecha:

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

ACELERACIÓN EN VHDL PARA FPGA DE ALGORITMOS DE DETECCIÓN DE BORDES (SOBEL / CANNY)

Autor: Martina Fraga Agras

Director: Fermín Zabalegui Sanz

Madrid

Junio de 2026

Agradecimientos

A mi director del proyecto, Fermín, por la dedicación, comprensión y aprendizaje brindados durante este último periodo.

A mis amigos, los de Madrid, por hacer de esta etapa una experiencia inolvidable, y los de Santiago, por haber escrito la historia conmigo.

A mis tíos y abuelos, por su cariño incondicional y por celebrar mis alegrías.

A mi hermano, por crecer conmigo y por ser la alegría de la casa.

A mis padres, por hacer esto posible, por ser la referencia a seguir y de la que aprender, y por darme la mejor herencia: una buena educación, valores y estudios.

A todos aquellos que de alguna manera han formado parte de este camino, gracias por acompañarme y hacerme crecer.

Martina Fraga
Madrid, 2026

ACELERACIÓN EN VHDL PARA FPGA DE ALGORITMOS DE DETECCIÓN DE BORDES (SOBEL / CANNY)

Autor: Fraga Agras, Martina.
Director: Zabalegui Sanz, Fermín.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

En este proyecto se han desarrollado los algoritmos de detección de bordes Sobel y Canny en VHDL y se han implementado en una FPGA. El proyecto tiene el objetivo de comparar esta implementación con una implementación convencional software en una Raspberry pi y demostrar que la FPGA es una alternativa competente. Se analizan costes energéticos, tiempo de detección y calidad de detección para realizar una comparativa entre ambas implementaciones.

Palabras clave: Sobel, Canny, bordes, VHDL, FPGA, hardware

1. Introducción y estado del arte

El procesamiento digital de imágenes está incrementando el área de aplicación en los últimos años debido a su capacidad para extraer información externa a través de las imágenes. La detección de bordes permite identificar cambios de intensidad en las imágenes, que se corresponden con los bordes de las imágenes. Esto facilita tareas posteriores como la segmentación, el reconocimiento de patrones o el análisis geométrico de las imágenes digitalmente.

Destacan los algoritmos de detección de bordes Sobel y Canny. En este ámbito las FPGA, circuitos de puertas lógicas programables, se proponen como una alternativa competente, capaz de explotar el paralelismo de cálculos con una gran eficiencia energética. [1]

Los algoritmos de detección de bordes Sobel y Canny parten de una imagen en escala de grises [2].

El algoritmo de detección de bordes **Sobel** se basa en una aproximación del cálculo del gradiente para la función discreta profundidad de color, en función de x e y. El gradiente se calcula en las dos direcciones [3]:

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

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

Utilizando esta aproximación se obtienen las matrices de convolución [3]:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Para llevar a cabo el cálculo se disponen los píxeles de la imagen en ventanas de 3x3 y, multiplicando cada elemento por el correspondiente en las matrices de convolución, se obtienen los gradientes en las direcciones x e y del píxel central. Se calcula el módulo del gradiente total:

$$|G| = \sqrt{{G_x}^2 + {G_y}^2} \sim |G_x| + |G_y|$$

A partir del establecimiento de un umbral y comparación del resultado del gradiente con el umbral se define cada píxel como borde o no borde.

Por otro lado, el algoritmo de detección de bordes **Canny** cuenta con más etapas. Inicialmente se aplica un suavizado gaussiano de la imagen, con la finalidad de eliminar el ruido, a través de aplicar la convolución a la imagen con la siguiente matriz de convolución normalizada, que sigue una distribución gaussiana [4]:

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Posteriormente se calcula el gradiente, de la misma forma que en Sobel y se aplica la supresión de no máximos: se pretende "adelgazar" los bordes para que tengan un ancho de 1 píxel. Para ello se eliminan los bordes que no sean máximos locales en la dirección del gradiente [5]:

$$\theta = tan^{-1}(\frac{G_y}{G_x})$$

Por último, a través del establecimiento de un doble umbral se definen los píxeles como bordes débiles, bordes fuertes y no bordes. A través de la etapa de histéresis se eliminan los bordes débiles independientes.

2. Definición del proyecto

La motivación principal del proyecto se basa en el análisis y demostración de las ventajas que las arquitecturas hardware ofrecen con respecto a las presentadas por las arquitecturas software. A través de este proyecto se requiere, no solo comprobar la viabilidad técnica de la aceleración hardware, sino también verificar que la implementación hardware de los filtros constituye una alternativa competente en términos de rendimiento y eficiencia, mediante la implementación simultánea de los filtros en un procesador de propósito general.

El objetivo general de este Trabajo de Fin de Grado es el diseño e implementación de un sistema de aceleración hardware en VHDL utilizando los algoritmos Sobel y Canny para detección de bordes para una FPGA Artix-7. A partir de este objetivo general, surgen otros objetivos más específicos:

- Diseñar e implementar en VHDL los sistemas Sobel y Canny
- Desarrollar una arquitectura pipeline que permita el procesamiento paralelo de píxeles
- Evaluar el rendimiento del sistema hardware

- Implementar los mismos sistemas en un dispositivo software y comparar las prestaciones

3. Descripción del sistema desarrollado en VHDL

3.1. Sobel

La estructura general simplificada del algoritmo Sobel en VHDL es la siguiente:

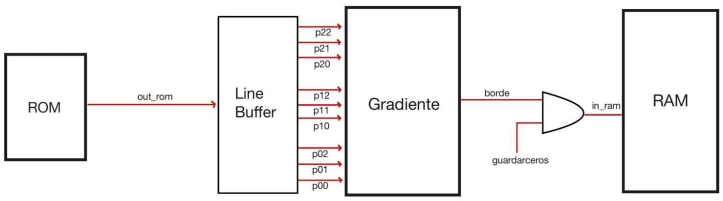


Ilustración 1. Estructura general Sobel

La estructura general del algoritmo Sobel está basada en un diseño pipeline, donde cada píxel va pasando por las etapas correspondientes del algoritmo sobel que se ha explicado. Los píxeles de la imagen, que se encuentran en la ROM, salen a una razón de uno cada ciclo de reloj. La función del linebuffer es disponer los píxeles en ventanas de 3x3 para hacer posible el posterior cálculo del gradiente del píxel central. En el bloque gradiente se calcula el gradiente del píxel central de la ventana 3x3 a través de las matrices de convolución y se compara el resultado con un umbral, impuesto desde el archivo superior de la jerarquía, para determinar si el resultado es un borde o no.
El objetivo de la señal guardarceros es guardar un 0 en la ram cuando en la salida del bloque gradiente se encuentre el resultado correspondiente a un píxel que es esquina de la imagen (primera y última fila y primera y última columna). Esto se debe a que para estos píxeles el gradiente está mal calculado, pues los píxeles esquina no tienen vecinos suficientes para completar la ventana 3x3. El control esta señal y de las etapas del sistema se realiza a través de un bloque de control, que se modela como una máquina de estados:

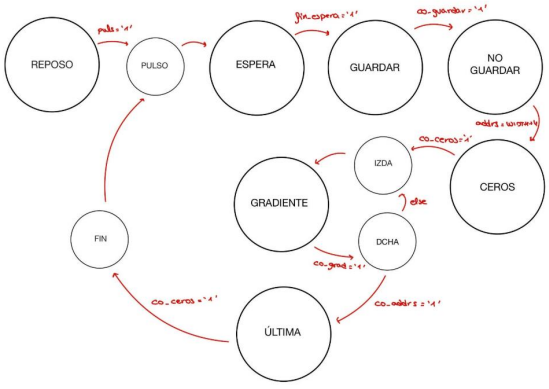


Ilustración 2. Máquina de estados control sobel

3.2. Canny

El algoritmo Canny tiene una estructura muy similar, pero incluye las etapas extras que se han explicado anteriormente:

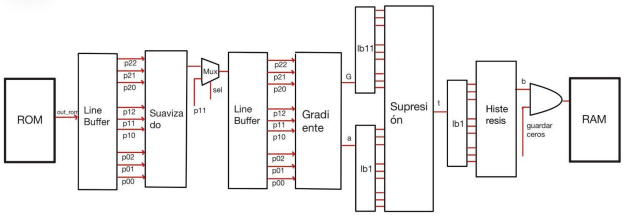


Ilustración 3. Estructura general Canny

El sistema también se basa en una arquitectura pipeline, donde los píxeles van pasando por las sucesivas etapas. En este caso, el bloque de control tiene que controlar dos señales: sel (cuando el píxel correspondiente a la salida del suavizado es un píxel esquina, el suavizado está mal calculado, por lo tanto, se cogerá el valor de profundidad de color original) y guardarceros, que ya es conocida.

4. Resultados

Una vez desarrollado el algoritmo la implementación se da en los siguientes escenarios, para permitir una comparativa:

- FPGA Artix-7 XC7A35T (placa Basys-3) a 100 MHz, con el pipeline VHDL que procesa un píxel por ciclo de reloj y aritmética entera.
- Raspberry Pi 4B (CPU ARM Cortex-A72 de cuatro núcleos a 1,5 GHz, 4 GB de RAM)

Para el análisis y comparación de los resultados las implementaciones tanto en software como en hardware tienen que ser idénticas, para realizar una comparación en las mismas condiciones. Es decir, la implementación software en la Raspberry pi es una imitación y resuelve exactamente el mismo problema que el código desarrollado en VHDL.

Las medidas para realizar la comparativa se han realizado para dos resoluciones:

- QVGA 320 x 240: A través de la implementación física en las placas.
- VGA 640 x 480: A través de una simulación.

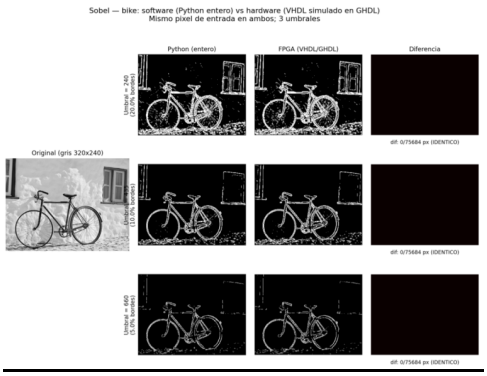
Se adjunta una tabla que resume los resultados relativos a la rapidez en la detección:

VGA640X480

Plataforma	Latencia/frame(ms)	FPS	Mpix/s
FPGA	3,09	326	100
Raspberry pi (1 núcleo)	125,93	7,94	2,44
Raspberry pi (4 núcleos)	90,5	11,05	3,39

Tabla 1: Resumen y comparativa características de rapidez en la detección FPGA y raspberry (VGA 640 x 480)

Uno de los resultados del mapa de bordes final, tanto en la FPGA como en Python, y la comparativa entre ambos es el siguiente:



Por último, se adjunta la tabla comparativa de los resultados obtenidos a nivel energético:

	P reposo (W)	**P carga (W)**	**J/frame**	**J/Mpix**
FPGA	0,5	0,666	0,002	0,0066
Raspberry Pi	3	6,4	0,58-0,82	1,89-2,67

Tabla 2: Comparación energética

5. Conclusiones

Las FPGA se presentan como una alternativa mejor en términos energéticos y competente en términos de rapidez y calidad de la detección.

6. Referencias

[1] S. Ravichandran, H.-K. Su, W.-K. Kuo, D. Dhanasekaran, M. Mahalingam y J.-P. Yang, «Parallel Processing of Sobel Edge Detection on FPGA: Enhancing Real-Time Image Analysis,» *Sensors,* vol. 25, 2025.

[2] Y. S. Domingo, «Universidad Politécnica de Valencia,» [En línea]. Available: https://yosedo.webs.upv.es/curso_edicion_de_imagenes_nivel_basico/1_7.html. [Último acceso: 7 Junio 2026].

[3] Departamento de Ingeniería electrónica de la Universidad de Jaén, «Detección de bordes en una imagen,» Jaén, 2006.

[4] J. Villemejane, «Lense Instititut Optique,» 2024. [En línea]. Available: https://iogs-lense-training.github.io/image-processing/contents/opencv_blur.html. [Último acceso: Junio 2026].

[5] J. V. Rebaza, «Detección de bordes mediante el algoritmo de Canny,» [En línea]. Available:https://www.researchgate.net/profile/Jorge-Valverde-Rebaza/publication/267240432_Deteccion_de_bordes_mediante_el_algoritmo_de_Canny/links/548dd1ae0cf225bf66a5f636/Deteccion-de-bordes-mediante-el-algoritmo-de-Canny.pdf. [Último acceso: Marzo 2026].

VHDL ACCELERATION FOR FPGA EDGE DETECTION ALGORITHMS (SOBEL / CANNY)

Author: Fraga Agras, Martina.
Supervisor: Zabalegui Sanz, Fermín.
Collaborating Entity: ICAI – Universidad Pontificia Comillas.

ABSTRACT

In this project, the Sobel and Canny edge detection algorithms were developed in VHDL and implemented on an FPGA. The project aims to compare this implementation with a conventional software implementation on a Raspberry Pi and demonstrate that the FPGA is a viable alternative. Energy costs, detection time, and detection quality are analysed to compare the two implementations.

Keywords: Sobel, Canny, edges, FPGA, VHDL, hardware

1. Introduction and state of art

Digital image processing has been expanding its application area in recent years due to its ability to extract external information from images. Edge detection allows the identification of intensity changes in images that correspond to image edges. This facilitates subsequent tasks such as segmentation, pattern recognition, and geometric analysis of digital images.

The Sobel and Canny edge detection algorithms are particularly noteworthy. In this field, FPGAs (Field-Profile Packet Arrays) are proposed as a viable alternative, capable of exploiting parallel processing and offering high energy efficiency. [1]

The Sobel and Canny edge detection algorithms start with a grayscale image [2].

The Sobel edge detection algorithm is based on an approximation of the gradient calculation for the discrete color level function, as a function of x and y. The gradient is calculated in both directions [3]:

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

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

Using this approach, the convolution matrices are obtained [3]:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

To perform the calculation, the image pixels are arranged in 3x3 windows, and by multiplying each element by its corresponding element in the convolution matrices, the gradients in the x and y directions of the central pixel are obtained. The magnitude of the total gradient is then calculated:

$$|G| = \sqrt{{G_x}^2 + {G_y}^2} \sim |G_x| + |G_y|$$

By establishing a threshold and comparing the gradient result with that threshold, each pixel is defined as an edge or not.

On the other hand, the Canny edge detection algorithm has more stages. Initially, Gaussian smoothing is applied to the image to eliminate noise by applying convolution to the image with the following normalized convolution matrix, which follows a Gaussian distribution [4]:

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

The gradient is then calculated, in the same way as in Sobel, and non-maximum suppression is applied: the aim is to "thin" the edges so that they have a width of 1 pixel. To do this, edges that are not local maxima in the direction of the gradient are eliminated [5]:

$$\theta = tan^{-1}(\frac{G_y}{G_x})$$

Finally, by establishing a double threshold, pixels are defined as weak edges, strong edges, and no edges. Independent weak edges are then removed through the hysteresis stage.

2. Project Definition

The main motivation for this project is based on the analysis and demonstration of the advantages that hardware architectures offer compared to those presented by software architectures. This project aims not only to verify the technical feasibility of hardware acceleration but also to demonstrate that the hardware implementation of filters constitutes a competitive alternative in terms of performance and efficiency, through the simultaneous implementation of the filters on a general-purpose processor.

The overall objective of this Thesis is the design and implementation of a hardware acceleration system in VHDL using the Sobel and Canny algorithms for edge detection on an Artix-7 FPGA. From this overall objective, several more specific objectives arise:

- Design and implement the Sobel and Canny systems in VHDL
- Develop a pipeline architecture that enables parallel pixel processing
- Evaluate the performance of the hardware system
- Implement the same systems on a software device and compare the performance

3. Description of the system developed in VHDL

3.1. Sobel

The simplified general structure of the Sobel algorithm in VHDL is as follows:

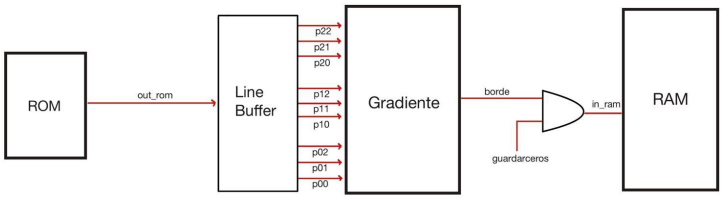


Figure 1. General Sobel structure

The general structure of the Sobel algorithm is based on a pipeline design, where each pixel passes through the corresponding stages of the algorithm as explained. The image pixels, which are stored in ROM, are output at a rate of one per clock cycle. The linebuffer's function is to arrange the pixels into 3x3 windows to enable the subsequent calculation of the gradient of the central pixel. In the gradient block, the gradient of the central pixel of the 3x3 window is calculated using convolution matrices, and the result is compared to a threshold, imposed from the file above in the hierarchy, to determine whether the result is an edge or not.

The purpose of the save-zero signal is to store a 0 in RAM when the gradient block outputs a result corresponding to a pixel that is a corner of the image (first and last row and first and last column). This is because the gradient is incorrectly calculated for these pixels, as the corner pixels do not have enough neighbours to fill the 3x3 window. The control of this signal and the stages of the system is carried out through a control block, which is modeled as a state machine:

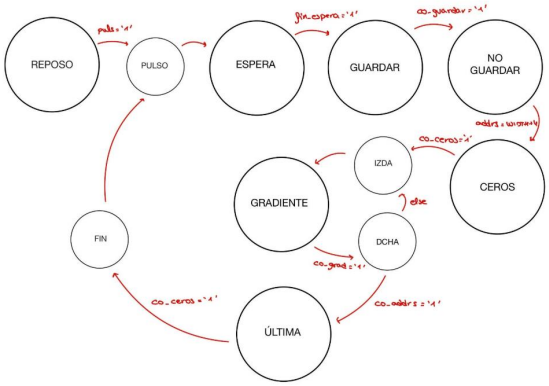


Figure 2. State machine control over

3.2. Canny

The Canny algorithm has a very similar structure, but includes the extra steps that have been explained above:

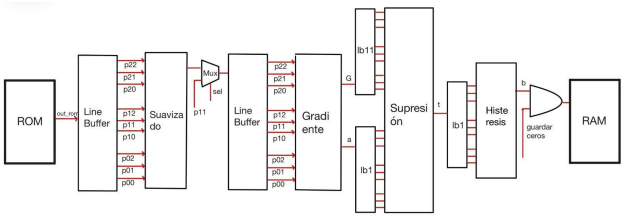


Figure 3. General structure of Canny

The system is also based on a pipeline architecture, where pixels pass through successive stages. In this case, the control block must manage two signals: sel (when the pixel corresponding to the smoothing output is a corner pixel, the smoothing is incorrectly calculated, therefore the original color depth value will be used) and save-zero, which is already known.

4. Results

Once the algorithm is developed, the implementation takes place in the following scenarios, to allow for comparison:

- FPGA Artix-7 XC7A35T (board Basys-3) a 100 MHz, with the VHDL pipeline that processes one pixel per clock cycle and integer arithmetic.
- Raspberry Pi 4B (Quad-core ARM Cortex-A72 CPU at 1.5 GHz, 4 GB of RAM)

For the analysis and comparison of results, both the software and hardware implementations must be identical to ensure a fair comparison. In other words, the software implementation on the Raspberry Pi is a replica and solves the exact same problem as the code developed in VHDL.

The measurements for this comparison were taken at two resolutions:

- QVGA 320 x 240: Through physical implementation on the motherboards.
- VGA 640 x 480: Through simulation.

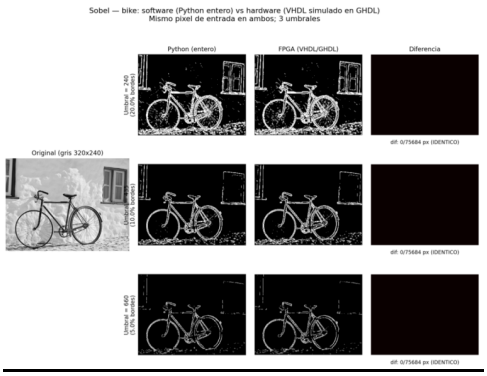
A table summarizing the detection speed results is attached.:

VGA640X480

Platform	Latency/frame(ms)	FPS	Mpix/s
FPGA	3,09	326	100
Raspberry pi (1 core)	125,93	7,94	2,44
Raspberry pi (4 cores)	90,5	11,05	3,39

Table 1: Summary and comparison of speed characteristics in FPGA and raspberry detection (VGA 640 x 480)

One of the results of the final edge map, both in the FPGA and in Python, and the comparison is as follows:



Finally, the comparative table of the results obtained at the energy level is attached.:

	P reposo (W)	P carga (W)	J/frame	J/Mpix
FPGA	0,5	0,666	0,002	0,0066
Raspberry Pi	3	6,4	0,58-0,82	1,89-2,67

Table 2: Energy Comparison

5. Conclusions

FPGAs are presented as a better alternative in terms of energy efficiency and are competent in terms of speed and quality of detection.

6. References

[1] S. Ravichandran, H.-K. Su, W.-K. Kuo, D. Dhanasekaran, M. Mahalingam y J.-P. Yang, «Parallel Processing of Sobel Edge Detection on FPGA: Enhancing Real-Time Image Analysis,» *Sensors,* vol. 25, 2025.

[2] Y. S. Domingo, «Universidad Politécnica de Valencia,» [En línea]. Available: https://yosedo.webs.upv.es/curso_edicion_de_imagenes_nivel_basico/1_7.html. [Last access: 7 Junio 2026].

[3] Departamento de Ingeniería electrónica de la Universidad de Jaén, «Detección de bordes en una imagen,» Jaén, 2006.

[4] J. Villemejane, «Lense Instititut Optique,» 2024. [En línea]. Available: https://iogs-lense-training.github.io/image-processing/contents/opencv_blur.html. [Last access: Junio 2026].

[5] J. V. Rebaza, «Detección de bordes mediante el algoritmo de Canny,» [En línea]. Available:https://www.researchgate.net/profile/Jorge-Valverde-Rebaza/publication/267240432_Deteccion_de_bordes_mediante_el_algoritmo_de_Canny/links/548dd1ae0cf225bf66a5f636/Deteccion-de-bordes-mediante-el-algoritmo-de-Canny.pdf. [Last access: Marzo 2026].

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN Y ESTADO DEL ARTE

1.1 INTRODUCCIÓN

El procesamiento digital de imágenes es una de las áreas más importantes de la ingeniería debido a su aplicación en sistemas industriales. La capacidad de extraer información externa de las imágenes es un método muy importante en algunos ámbitos: medicina industrial, conducción autónoma, automatizaciones industriales, robótica, etc.

Más en concreto, la detección de bordes es una función fundamental ya que permite identificar relieves y cambios bruscos de intensidad en las imágenes, que se corresponden con los contornos. Esta tecnología ha hecho posibles muchos avances en distintas áreas destacando aplicaciones en medicina, pues, tal y como se muestra en la *Figura 1,* permite la detección eficaz de tumores, análisis de volúmenes y formas de algunos órganos, o detección de enfermedades como la hipertensión a través del análisis de los vasos sanguíneos oculares, entre otros. Otro de los avances que ha proporcionado esta tecnología es la detección de vías y carreteras. Esta funcionalidad se incluye en la mayoría de los sistemas avanzados de ayuda a la conducción (ADAS), que utilizan la detección de bordes para detectar la posición del vehículo en la carretera, y advertir al conductor cuando es necesario [1]. Además, la capacidad de analizar la vía y detectar los bordes de la carretera ha sido una tecnología clave en el desarrollo de la conducción autónoma.

La detección de bordes facilita otras tareas como pueden ser la segmentación, el reconocimiento de patrones o el análisis geométrico de las imágenes. Estas funciones permiten acciones más avanzadas, como la detección y comprensión de las señales de tráfico, para la ayuda en la conducción o conducción autónoma, reconocimiento facial, para la tecnología móvil o la detección de defectos y micro fisuras en la cadena de montaje y producción de productos.

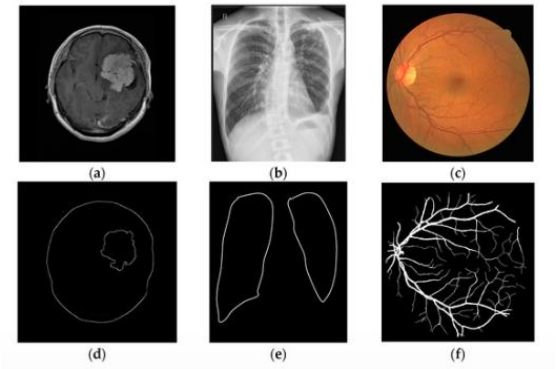


Figura 1. Aplicaciones de la detección de bordes en medicina [2]

En este campo destacan las herramientas Sobel y Canny, filtros que permiten la detección de bordes y que son muy utilizados en entornos académicos e industriales debido a su eficacia y robustez frente al ruido. Cada uno de estos filtros ofrece ciertas ventajas y desventajas. El operador Sobel se basa en el cálculo del gradiente de una imagen mediante matrices de convolución. Por otro lado, el algoritmo Canny es más complejo, ya que requiere varias etapas (suavizado, cálculo de gradientes, supresión de no máximos y doble umbral) lo que proporciona una mayor robustez y eficacia en la detección. El detector de bordes Canny es uno de los métodos más robustos, capaz de reducir el ruido. Sin embargo, la complejidad de este filtro en comparación con el algoritmo Sobel suele hacer que este último sea preferible. El operador Sobel se basa en el cálculo del gradiente de una imagen mediante matrices de convolución. Existen también otros métodos de detección de bordes como los algoritmos Pewitt, Roberts o el laplaciano, que no serán objeto de estudio en este trabajo, pero que también tienen sus ventajas. [3]

Tradicionalmente, el tratamiento de imágenes necesitaba muchos datos, memoria y registros, que no encajaban por tamaño y velocidad en hardware programable y la única alternativa era implementación software. Con la mejora de la velocidad y capacidad de las FPGAs, actualmente sí que se consideran una alternativa para trabajar en tiempo real que, a diferencia de la implementación en procesadores, permite realizar cálculos en paralelo y se proponen como una solución competente. [3]

Las FPGA (Field Programmable Gate Arrays) pueden constituir una plataforma ideal para la implementación de sistemas de visión artificial. Debido a su capacidad reconfigurable y la posibilidad que ofrecen de definir estructuras hardware específicas, permiten explotar el paralelismo que requieren los algoritmos de detección de bordes, especialmente Sobel y Canny que necesitan un elevado número de operaciones de convolución por píxel y se benefician significativamente de este paralelismo en el cálculo.

El presente proyecto abordará el diseño en VHDL de un sistema de detección de bordes para FPGA, basado en los filtros Sobel y Canny. Se trabajará sobre imágenes obtenidas a partir de una fuente de video o cámara, procesándolas en tiempo real y generando un mapa de bordes.

1.2 ESTADO DEL ARTE

El campo de la implementación de algoritmos de procesamiento de imágenes y detección de bordes en el entorno de las FPGA o sistemas hardware ha sido motivo de estudio en los últimos años debido a la necesidad de mejorar la eficiencia de la implementación tradicional software. Entre los métodos más utilizados se encuentran los algoritmos Sobel y Canny, que han sido analizados tanto desde el punto de vista teórico como práctico debido a su alto rendimiento y sus altas capacidades para eliminar el ruido.

1.2.1 ALGORITMO SOBEL

El algoritmo Sobel es un algoritmo simple, que consta de una sola etapa: la asignación de cada píxel en borde o no borde mediante el cálculo del gradiente en las dos direcciones y la imposición de un umbral. Se parte de una imagen en escala de grises, esto es, cada píxel tiene una intensidad de color comprendida entre 0 y $2^N - 1$, siendo el 0 el negro puro (nada de brillo) y el $2^N - 1$ el blanco puro (máximo brillo) [4]. Con estos valores de intensidad en escala de grises se calcula el gradiente en dos direcciones mediante la convolución de ventanas de 3x3 píxeles. El procedimiento se basa en una aproximación de la derivada para

funciones discretas (se tiene un único valor de intensidad de color para cada píxel, no es un valor continuo. Es por ello por lo que hace falta una aproximación para funciones discretas). Según la *Ecuación 1,* una manera de aproximar la derivada de una función (intensidad de color) en x o en y (las dos direcciones de la imagen), es restando los valores del píxel siguiente y el anterior en esa dirección. El filtro Sobel para el cálculo del gradiente dispone los pixeles en ventanas de 3x3, calculando así el gradiente del píxel central mediante las matrices Kernel de referencia que se observan en la *Figura 2.* El procedimiento consiste en multiplicar cada elemento de la matriz 3x3 de intensidades de color de los píxeles, por su elemento correspondiente en la matriz Kernel. De esta forma se utiliza la aproximación comentada y además es posible ponderar los píxeles con simplemente cambiar los números de cada elemento de la matriz Kernel.

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

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

Ecuación 1. [5]

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Figura 2. Matrices de convolución [5]

Utilizando el procedimiento explicado se obtienen los valores de los gradientes en la dirección horizontal, x, y vertical, y, de la imagen. Para el cálculo del módulo del gradiente se utiliza la *Ecuación 2* aproximándolo a la suma de los valores absolutos de los gradientes en ambas direcciones.

$$|G| = \sqrt{{G_x}^2 + {G_y}^2} \sim |G_x| + |G_y|$$

Ecuación 2

Una vez calculada la magnitud del gradiente del píxel central de la ventana, se establece un umbral para el cual si:

G<U →No borde

G>U →Borde

El operador Sobel, ha sido estudiado en diversos trabajos implementándolo sobre una FPGA. Un ejemplo de ello es el estudio [6] en el que se implementa un sistema de detección de bordes multidireccional Sobel, utilizando el lenguaje de alto nivel VHDL sobre una FPGA para detectar relieves sobre imágenes 1024 x 1024. En este trabajo se propone una arquitectura hardware para el cálculo de bordes en 4 direcciones: vertical y horizontal, como habitualmente, y adicionalmente en las dos diagonales. Esta capacidad de cálculo en 4 direcciones amplia la capacidad del operador Sobel y optimiza el paralelismo característico de los sistemas hardware. Se incluyen imágenes con los resultados de aplicar el operador Sobel con la finalidad de apreciar su eficacia. El trabajo realizado por Roberto Millon y Emmanuel Frati [7] explora también, como alternativa a la implementación de VHDL sobre FPGA, otro lenguaje de alto nivel (HLS). Este estudio revela la viabilidad de implementar filtros de detección de bordes sobre plataformas SoC FPGA con herramientas de síntesis de alto nivel. Ambos estudios demuestran la eficacia y simplicidad del operador Sobel, capaz de procesar imágenes en tiempo real, detectando los bordes con alta precisión.

1.2.2 ALGORITMO CANNY

Por otro lado, el algoritmo Canny sigue un proceso más complejo que incluye varias etapas. Inicialmente, antes de llevar a cabo la detección de bordes se elimina el ruido de la imagen a través de un **suavizado gaussiano**. Este suavizado se realiza por convolución de ventanas de 3x3 píxeles a través de la matriz de referencia que se muestra en la *Figura 3.* Esta matriz es el resultado de aplicar una distribución gaussiana *Ecuación 3.* para conseguir los pesos de cada uno de los elementos de la ventana. En una distribución gaussiana, se ponderan con un número más elevado las intensidades de color originales del píxel central y de los píxeles adyacentes, mientras que los píxeles de las esquinas (que tienen una distancia mayor respecto al píxel central) se tienen menos en cuenta. Dividir el resultado entre 16 es una cuestión de

normalización, para que el nuevo valor de intensidad de color tenga el mismo rango que el original [8]. Es habitual encontrar matrices de referencia para el suavizado gaussiano diferentes, incluso en algunos casos, la convolución se hace con ventanas de píxeles más grandes [9], pero en todos los casos se sigue la misma lógica de ponderación, siendo los píxeles centrales y adyacentes más importantes (propio de una distribución gaussiana), teniendo siempre cuidado de no realizar un suavizado excesivo que provoque la pérdida de detalles importantes de la imagen y por consecuencia, el fallo del algoritmo.

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

Figura 3. Matriz de Suavizado Gaussiano [8]

$$G(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2+(y-\mu)^2}{2\sigma^2}}$$

Ecuación 3. [8]

Una vez se tiene la nueva imagen, con los nuevos valores de intensidad de color de todos los píxeles, después de haber eliminado el ruido, se calcula el **gradiente** [10] con un procedimiento idéntico al explicado en el algoritmo Sobel. En este caso, es necesario calcular un parámetro extra, la dirección del gradiente, pues será necesaria para etapas posteriores. El cálculo de esta dirección se calcula con la *Ecuación 4*.

$$\theta = tan^{-1}(\frac{G_y}{G_x})$$

Ecuación 4 [9]

En ocasiones [11] los dos procesos anteriores se realizan a la vez y, en lugar de primero aplicar el filtro y después derivar el resultado *Ecuación 5*, como se ha explicado antes con sendas aproximaciones, se aprovecha la propiedad matemática *Ecuación 6* y se deriva la función de distribución gaussiana *Ecuación 3.* para posteriormente multiplicar el resultado por I(x,y) (valor de intensidad de color original de los píxeles de la imagen).

$$I_G = I(x,y) * G(x,y)$$

$$G_x = \frac{dI_G}{dx}, \quad G_y = \frac{dI_G}{dy}$$

Ecuación 5

$$\frac{dI}{dx} * G = \frac{dG}{dx} * I$$

Ecuación 6

El siguiente paso es la **supresión de no máximos** [10]. El objetivo de esta etapa es "adelgazar" los bordes, para que solo ocupen 1 píxel. Para ello se analiza la dirección del gradiente θ y se compara la magnitud del píxel estudiado con los adyacentes en la dirección de θ. Si el píxel no tiene el valor del gradiente máximo local, se suprime, y no será contabilizado como borde. Los valores válidos para θ serán 0º, 45º, 90º y 135º, pues son las únicas direcciones en las que existen píxeles vecinos. De esta forma, θ tendrá que aproximarse a uno de esos valores.

Por último, se lleva a cabo un proceso de **histéresis mediante doble umbral** [9]. En esta etapa se establecen dos umbrales, uno alto y uno bajo (Th, Tl). Se clasifican los bordes entre fuertes y débiles:

G>Th→Borde fuerte

Tl<G<Th→Borde débil

G<Tl→No borde.

Los bordes débiles que no son adyacentes con un borde fuerte se eliminan. Así, obtenemos como bordes únicamente los bordes fuertes y los bordes débiles que no están "sueltos".

Se han llevado a cabo estudios que reflejan la efectividad de la implantación del sistema de detección de bordes Canny para FPGA. En este ámbito, Jorge Valverde Rebaza presenta un estudio detallado sobre este algoritmo [9]. En su trabajo, Valverde, recoge las principales etapas de la aplicación del filtro. En [9] también se detallan los criterios de detección, localización y respuesta única, definidos por Canny en la formulación original. El estudio

incluye también la aplicación del filtro sobre imágenes reales, y la exposición de los resultados obtenidos, mostrando como cada etapa contribuye a la calidad de la detección y mostrando la combinación entre la robustez frente al ruido y la eficacia en la precisión de detección.

En la literatura se encuentran también estudios orientados a la comparación y rendimiento de los diferentes algoritmos de detección. Un ejemplo de ello es [12], que analiza la implementación de múltiples sistemas de detección (Roberts, Sobel, Prewitt, Canny) sobre una misma FPGA para evaluar la calidad de detección y la complejidad del algoritmo. Dentro del estudio de la comparativa entre filtros, existen trabajos [13] [14] que ofrecen una comparativa más directa, estudiando únicamente los filtros Sobel y Canny. En el estudio [13]se evalúan aspectos como la precisión, la sensibilidad al ruido y el coste computacional y complejidad asociada a cada uno de los algoritmos llevando a cabo una implementación software de estos. Por su parte, [14] utiliza FPGA para la comparación de los filtros. Ambos estudios reflejan que Canny presenta una detección de bordes más precisa y continua, y sobre todo en presencia de ruido, debido a sus múltiples etapas de procesamiento. Sin embargo, se destaca que el operador Sobel tiene una menor complejidad computacional y mayor velocidad de ejecución, lo que hace que este filtro sea la opción más adecuada cuando se trata de procesamiento en tiempo real.

1.3 ESTRUCTURA DEL DOCUMENTO

Este documento empieza con este capítulo, donde se ha introducido el procesamiento digital de imágenes y la detección de bordes y se ha presentado un breve estudio de otros trabajos y avances realizados en referencia a los algoritmos Sobel y Canny. Se ha presentado una breve explicación teórica de ambos filtros. En el siguiente capítulo se detallan las herramientas utilizadas para el desarrollo del proyecto, tanto el desarrollo software como hardware.

En el capítulo 3 se definen los objetivos del trabajo, los principales motivos de la realización del proyecto, porqué es necesario y qué es lo que se intenta resolver. Además, se dará una breve explicación de la metodología a seguir y de la planificación.

En los siguientes capítulos se detalla el desarrollo del sistema y los análisis de los resultados obtenidos. Además, se incluye un capítulo que contiene las conclusiones y destaca lo que se ha hecho, deja claros qué objetivos se han cubierto y cuáles son las aportaciones hechas con el fin de una mayor claridad de cara a futuros trabajos.

Al final del documento se incluyen los anexos. En ellos se incluye la alineación del trabajo con los Objetivos de Desarrollo Sostenible y el código fuente necesario para el desarrollo hardware.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

En esta sección se describirán las tecnologías y herramientas utilizadas para el correcto desarrollo del trabajo.

2.1 HERRAMIENTAS PARA LA IMPLEMENTACIÓN SOFTWARE

Para el desarrollo software se realizará en Python mediante el editor de código Visual Studio Code.

2.1.1 PYTHON

Python es un lenguaje de programación de alto nivel interpretable y multipropósito. Es un lenguaje orientado a objetos y es ampliamente utilizado en el ámbito científico debido a la su sintaxis sencilla y facilidad para entender y aprender por un humano. Esta característica reduce el tiempo mantenimiento de los sistemas, así como su coste [15]. Además, la facilidad y rapidez del ciclo de edición de Python, al no necesitar compilación, lo convierte en una herramienta clave para el desarrollo de algoritmos de detección de bordes, pues esta característica agiliza el proceso de desarrollo y el proceso de pruebas y verificación [15]. Otra de las virtudes de Python es que es un lenguaje multiplataforma [15], es decir, se puede implementar en casi cualquier sistema operativo sin tener que realizar apenas cambios en el código. Python dispone de numerosas librerías [15], muchas de ellas dedicadas al procesamiento de imágenes, lo que lo hace un lenguaje ideal para el desarrollo de este proyecto.

Figura 4. Logo de Python [16]

2.1.2 VISUAL STUDIO CODE

Figura 5. Logo de VS Code

Visual Studio Code (VS Code) es un editor de código de código abierto desarrollado por Microsoft que permite la edición de múltiples lenguajes de programación (Python, C/C++, HTML, JavaScript…). Para todos ellos VS Code ofrece sistemas de resaltado de sintaxis y sugerencias de autocompletado, además de compiladores [17]

Visual Studio Code es un editor de texto multiplataforma, que está disponible tanto en Windows, Linux, macOS… [17]. Además, una característica que lo desmarca del resto de editores de código es que cuenta con un número elevado de extensiones que permite a los usuarios agregar funcionalidades [17].

VS Code integra también herramientas para trabajar con repositorios de Git directamente desde la interfaz gráfica [17]. Esto facilita el seguimiento de los cambios realizados y permite recuperar versiones anteriores del código.

Por todo ello se considera una herramienta ideal para ser utilizada en este proyecto para la implementación de los algoritmos de detección de bordes en software.

2.2 HERRAMIENTAS PARA LA IMPLEMENTACIÓN HARDWARE

Para el desarrollo en hardware del sistema se ha elegido una FPGA Artix-7 como plataforma para la implementación. La descripción del hardware se ha hecho en VHDL a través del entorno de desarrollo Vivado en su versión 2025.2.

2.2.1 VHDL

VHDL es un lenguaje de descripción de hardware desarrollado a mediados de los años 80 por el IEEE y el departamento de defensa de los Estados Unidos de América [18]. Describe la estructura y el comportamiento de circuitos electrónicos. La utilidad de un lenguaje de especificación hardware es poder construir descripciones de circuitos que sean inequívocas [18]. VHDL permite describir hardware a distintos niveles de abstracción, desde puertas lógicas hasta sistemas más complejos con un nivel más alto.

Uno de los mayores avances ha sido el desarrollo de sintetizadores, que a partir de una especificación en VHDL son capaces de generar un circuito [18].

VHDL funciona mediante un diseño de jerarquías, donde, a medida que bajamos en la jerarquía se tienen bloques más simples. En lo alto de la jerarquía nos encontramos con bloques estructurales, que combinan y conectan los bloques más simples. Esto permite al ingeniero dividir el trabajo, pudiendo simularse por separado, siendo así la detección de errores más rápida y sencilla. Además, si se diseñan bloques lo suficientemente genéricos, estos pueden ser utilizados en otras partes del proyecto o incluso en otros proyectos [18].

2.2.2 VIVADO

Vivado Design Suite es un entorno de desarrollo de AMD-Xinlix que permite diseñar, simular, sintetizar e implementar sistemas digitales sobre FPGA y SoC programables [19].

Vivado cuenta con un flujo de diseño que permite transformar la descripción en VHDL del sistema en un circuito físico sobre una FPGA. Además de contar con correctores de texto y compiladores, que comprueban la correcta sintaxis del diseño (esto no asegura el correcto funcionamiento lógico), cuenta con simuladores mediante testbenches personalizables, que permiten estimular los bloques desarrollados con entradas que cambian de valor en los momentos que el usuario considere oportuno y comprobar el correcto funcionamiento de las salidas mediante un panel de simulación en el tiempo. Vivado incluye también sintetizadores que convierten el código en un circuito físico (Place and Route). Cuenta también con el

sistema de implementación y adaptación del circuito sobre una FPGA concreta, generando un archivo .bit que contiene la configuración completa de la FPGA.

Figura 6. Logo Vivado [20]

2.3 FUENTES DE INFORMACIÓN

Por último, para el estudio teórico y apoyo en la redacción de este documento, de han consultado archivos bibliográficos, libros, sitios web, artículos, etc. Las fuentes de información que se han usado se recogen en la Bibliografía de esta memoria.

Capítulo 3. DEFINICIÓN DEL TRABAJO

3.1 MOTIVACIÓN

En el contexto actual cada vez existen más sistemas que procesan vídeo en tiempo real: vehículos autónomos, drones, sistemas de vigilancia, inspección industrial, etc. El crecimiento en el uso de estos sistemas ha hecho necesaria la capacidad de procesado de imágenes en tiempo real, que ofrezca altas prestaciones de precisión, baja latencia y que haga un uso eficiente de los recursos disponibles. Las herramientas de detección de bordes, como se ha comentado en apartados anteriores, constituyen una etapa fundamental dentro de muchos de los procesos industriales, actuando como la base para el proceso de análisis posteriores más complejos.

Los algoritmos Sobel y Canny, así como otros sistemas de detección de bordes, han sido tradicionalmente estudiados e implementados en entornos software. Aunque el funcionamiento de estos sistemas es ejemplar, el consumo energético supone un problema cuando es necesaria la detección en tiempo real de imágenes con elevada resolución [21]. Es por ello por lo que existe la necesidad de explorar soluciones que permitan incrementar la eficiencia de este tipo de algoritmos sin comprometer la calidad y rapidez de los resultados.

En este escenario, las FPGA se presentan como la mejor alternativa para mejorar la eficiencia de estos sistemas y proponer soluciones de bajo coste energético para sistemas embebidos, pues, como ya se ha comentado anteriormente, son sistemas de hardware reconfigurable, que permiten explotar el paralelismo de cálculo que necesitan los algoritmos. No obstante, a pesar de las claras ventajas que ofrece, la aplicación de FPGA implica retos significativos, principalmente, la optimización del hardware y de los recursos disponibles y la implementación eficiente de arquitecturas pipeline que permitan cumplir los objetivos con la menor latencia posible. Además, el desarrollo de los algoritmos de detección de bordes

cuenta con un incremento de complejidad cuando se trata de la implementación en un lenguaje de descripción de hardware (como VHDL).

La motivación principal del proyecto, por lo tanto, se basa en el análisis y demostración de las ventajas que las arquitecturas hardware ofrecen con respecto a las presentadas por las arquitecturas software. Se procederá mediante el diseño y aplicación en VHDL de un sistema de detección de bordes, utilizando los algoritmos Sobel y Canny, sobre FPGA. A través de este proyecto se requiere, no solo comprobar la viabilidad técnica de la aceleración hardware, sino también verificar que la implementación hardware de los filtros constituye una alternativa competente en términos de rendimiento y eficiencia, mediante la implantación simultánea de los filtros en un procesador de propósito general.

Finalmente, este proyecto permite integrar todo lo aprendido en el Grado en Ingeniería en Tecnologías Industriales en áreas de procesamiento digital, procesamiento de señales, arquitectura de los computadores, fundamentos de informática y microprocesadores. De esta forma, se presenta como una oportunidad para aplicar de forma práctica los conocimientos adquiridos en un caso realista, que está alineado con las tendencias de investigación actuales.

3.2 OBJETIVOS

El objetivo general de este Trabajo de Fin de Grado es el diseño e implementación de un sistema de aceleración hardware en VHDL utilizando los algoritmos Sobel y Canny para detección de bordes para una FPGA Artix-7. Esta implementación tiene la finalidad de demostrar que los sistemas hardware son una efectiva solución frente a las limitaciones de los sistemas software en el procesado de imágenes y detección de bordes en tiempo real. Se implementarán también los filtros en un sistema software (Python) y se compararán las características de cada uno, teniendo como objetivo el trabajo, demostrar que el entorno hardware es más adecuado en este ámbito.

A partir de este objetivo general, surgen otros objetivos más específicos:

1. **Diseñar e implementar en VHDL los sistemas Sobel y Canny:** Se estudiarán y reorganizarán las estructuras de los algoritmos de detección Sobel y Canny, para poderlos implementar en una FPGA Artix-7 y para que sea posible la ejecución en paralelo. Se detallará el código VHDL necesario para el correcto funcionamiento.

2. **Desarrollar una arquitectura pipeline que permita el procesamiento paralelo de píxeles:** Se dividirá el proceso en etapas consecutivas, que permitan el procesamiento simultáneo de píxeles. Este objetivo incluye el tratamiento de los retardos introducidos por cada bloque, la sincronización y gestión de señales y flujos de datos entre etapas y la creación de una estructura consistente, que sea capaz de lograr un procesamiento continuo, estable y en tiempo real.

3. **Diseñar el sistema de salida de imagen:** Se proporcionará como salida el correspondiente mapa de bordes generado por los algoritmos y se incluirá su visualización mediante una interfaz VGA.

4. **Evaluar el rendimiento del sistema hardware:** Se darán algunos ejemplos, con imágenes de prueba, sobre el trabajo realizado por los filtros en un sistema hardware.

5. **Implementar los mismos sistemas en un dispositivo software y comparar las prestaciones:** Se diseñarán los mismos filtros Sobel y Canny para ser implementados en un procesador convencional. De esta forma, se compararán los resultados con los obtenidos a partir de la FPGA y se cuantificarán las ventajas y desventajas resultantes de llevar a cabo este tipo de procesamiento de imágenes en un entorno hardware.

3.3 METODOLOGÍA Y PLANIFICACIÓN

El proyecto se abordará dividiéndolo en secciones de estudio, que se realizarán una a continuación de la otra. Con el objetivo final de realizar correctamente la comparación de los filtros en software y hardware, el trabajo se dividirá en las siguientes secciones:

1. **Análisis requisitos y análisis teórico Sobel y Canny:** Consiste en la búsqueda de información y el análisis detallado de los algoritmos de detección Sobel y Canny, entendiendo todos los conceptos, su formulación matemática, sus etapas… La realización del Anexo B se incluye en esta etapa, pues este incluye el análisis de los requisitos y objetivos y la búsqueda de información. Esto permitirá trabajar con una mayor fluidez posteriormente, que serán necesarios estos conocimientos.

2. **Implementación software:** En esta segunda etapa se implementan los filtros Sobel y Canny en Python. Esta implementación permitirá el análisis de los resultados obtenidos sobre imágenes de prueba, una primera comparación entre los dos filtros, Sobel y Canny, y validar sus comportamientos.

3. **Diseño y simulación de los componentes en VHDL:** Se diseñan cada uno de los componentes (buffers, cálculos del gradiente, etapas de Canny…) de ambos filtros en VHDL y se simula para comprobar el funcionamiento individual de cada uno de estos componentes funcionales. Esto hará más llevadera la conformación final de los filtros, y permite identificar errores más fácilmente.

4. **Diseño de la jerarquía global de los filtros en VHDL:** Una vez diseñados todos los componentes necesarios se diseña la arquitectura global que incluye estos componentes funcionando conjuntamente. Se simulan ambos filtros y se comprueban si los resultados son los correctos antes de implementarlos físicamente. En esta etapa se analizarán las dimensiones de los diseños dentro de la FPGA y, según el caso se analizarán diferentes alternativas de realización, por si fuese necesario un diseño más secuencial para el ahorro de hardware.

5. **Implementación física:** Cuando se tienen los dos filtros diseñados y funcionando correctamente se implementan físicamente en la FPGA y se analiza si el comportamiento es el adecuado.

6. **Mejoras de la implementación:** Se irán añadiendo mejoras a lo largo del desarrollo del trabajo, como por ejemplo la obtención de imágenes a través de una cámara en tiempo real mediante bitstream o la representación del resultado de mapa de bordes mediante una VGA.

7. **Análisis de los resultados y comparación:** Finalmente, con los resultados obtenidos, se analizará detalladamente ambos métodos y se llevará a cabo una rigurosa comparación, analizando las ventajas de la implementación software. Esta comparación se incluirá en la memoria final, que se irá rellenando a medida que se desarrolla el trabajo.

	1.Estudio teórico	**2.Imp. software**	**3.Compon. VHDL**	**4.Jerarq. global**	**5.Imp. física**	**6. Mejoras**	**7.Análisis y comparación**
12-18ene							
19-25ene							
26-1feb							
2-8feb							
9-15feb							
16-22feb							
23-1mar							
2-8mar							
9-15mar							
16-22mar							
23-29mar							
30-5abr							
6-12abr							
13-19abr							
20-26abr							
27-3may							
4-10may							
11-17may							
18-24may							

Tabla 1. Planificación del proyecto

Capítulo 4. SISTEMA HARDWARE DESARROLLADO

4.1 ANÁLISIS DEL SISTEMA

El sistema desarrollado es un sistema de detección de bordes de imágenes de resolución 640 x 480 con profundidad de color de 8 bits. Es decir, se tiene un total de 256 valores de intensidad de color diferentes. Aunque el sistema esté diseñado para la detección de imágenes 640 x 480 es fácilmente escalable y aplicable para imágenes de diferente resolución, pues este parámetro se ha definido como un generic dentro de los bloques desarrollados en VHDL, lo que permite el cambio de escala con simplemente cambiar un par de líneas de código donde se define este parámetro. Se ha desarrollado tanto el algoritmo de detección de bordes Sobel como Canny, pero ambos tienen la misma función y funcionan de la misma manera, por lo que esta sección es una descripción del funcionamiento de ambos.

El sistema cuenta también con la funcionalidad necesaria para la conexión con una cámara externa, por si en un futuro surge la necesidad de realizar la detección en tiempo real a través de una cámara de vídeo. En el caso de esta conexión, el sistema funciona al conectarse a la cámara de video OV 7670, que capta constantemente imágenes del entorno. El algoritmo de detección de bordes se sincroniza con la cámara intercambiando señales que confirman la disponibilidad de uno y de otro (cámara y sistema) para enviar y recibir información de las imágenes. El resultado final se dispone en una interfaz digital VGA. Cuando se han detectado completamente los bordes de la imagen el sistema le pide automáticamente otra imagen a la cámara y repite el proceso. Cada una de las imágenes se procesa en milisegundos, por lo que, se está dando una nueva detección cada muy poco tiempo.

4.1.1 SINCRONIZACIÓN CON LA CÁMARA

El sistema de sincronización con la cámara es sencillo. La cámara necesita recibir una señal (un pulso) para comprender que se necesita capturar una nueva imagen. Es decir, cuando el

sistema esté listo para una nueva detección, debe enviarle un pulso a la cámara. Una vez recibida esta señal la cámara comienza con la captura de la imagen, mientras tanto, el sistema está en reposo, esperando a que la cámara acabe de capturar la imagen. Cuando se tiene toda la información necesaria de los píxeles de la imagen, se envía una señal (un pulso) desde la cámara al sistema para indicar que ya se tiene la imagen, y que se va a proceder con el envío de valores de intensidad de los píxeles, por orden, desde el píxel izquierdo superior al derecho inferior, fila por fila. El sistema recibe este pulso y se prepara para recibir los píxeles y guardarlos en una ROM. Los píxeles se envían uno a continuación del otro, en serie. Se envía uno cada ciclo de reloj. El paso de los 8 bits de cada píxel se realiza en paralelo.

4.2 DISEÑO SOBEL

En este apartado se detallará el desarrollo en VHDL del algoritmo de detección de bordes Sobel, así como su simulación en el tiempo mediante la herramienta Vivado 2025.2.

4.2.1 ESTRUCTURA GENERAL

La estructura general del algoritmo Sobel está basada en un diseño pipeline, donde cada píxel va pasando por las etapas correspondientes del algoritmo sobel que se ha explicado en el estado del arte del presente documento.

La *Figura 7* refleja la arquitectura que los píxeles de la imagen deben seguir. Este esquema de la arquitectura es un sistema simplificado, que no incluye los bloques de control ni los contadores necesarios, si no únicamente los bloques que intervienen directamente en el algoritmo Sobel con el fin de una mejor comprensión inicial de la estructura. En secciones posteriores se encuentra el esquema general completo del proyecto en VHDL: *Figura 18*.

Inicialmente, cuando se tiene la imagen ya guardada en la memoria, los píxeles de la imagen, que se encuentran en la ROM, salen a una razón de uno cada ciclo de reloj (el sistema de salida de los píxeles se realiza a través de un input que recibe la ROM (addrs, proporcionado por un bloque de control que se explicará más tarde en este documento, que contiene el lugar de la memoria donde se encuentra el píxel solicitado). Estos van entrando, uno a uno, a un

linebuffer. La función del linebuffer es disponer los píxeles en ventanas de 3x3 para hacer posible el posterior cálculo del gradiente del píxel central.

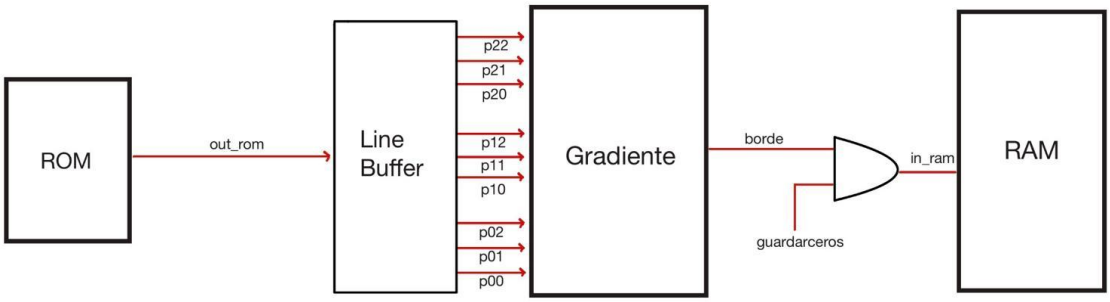


Figura 7. Estructura general simplificada de Sobel en VHDL

El valor final del gradiente de cada píxel es evaluado y comparado con un umbral donde se decide si el píxel se considera borde (1) o no borde (0). El valor final (0 o 1) de cada píxel se introduce en una puerta and. La otra entrada de esta puerta será una señal (guardarceros) que proviene de un bloque de control posteriormente explicado. Esta señal pone su valor a 0 cuando la salida del bloque gradiente se corresponde con un píxel esquina (los de la primera o última fila y los de la primera o última columna). Esto se debe a que el cálculo del gradiente de estos píxeles puede estar mal calculado ya que, los píxeles esquina, no tienen píxeles vecinos para poder hacer la ventana 3x3 a su alrededor (recuérdese que el cálculo del gradiente en una ventana 3x3 se realiza para el píxel central. Un píxel esquina no puede ser nunca el píxel central de una ventana 3x3). Es por ello por lo que estos píxeles se considerarán siempre como no bordes (guardarceros='0'). Una vez conocido el valor final del píxel (borde o no borde) se guarda uno a uno en la RAM. Posteriormente se explicará el conexionado de este sistema con el exterior (cámara y VGA).

4.2.2 ROM.VHD

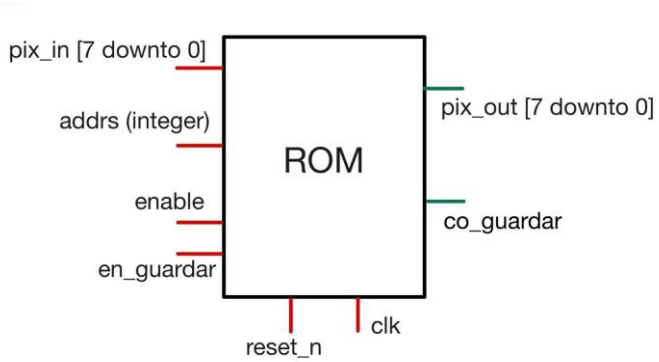


Figura 8. Bloque rom.vhd en VHDL

Este bloque tiene que recibir y almacenar píxeles cuando la cámara esté transmitiendo información y tiene que ser capaz de proporcionar esta información al sistema para la detección de bordes. Es decir, tiene dos métodos de funcionamiento: guardando (la información que se recibe por la cámara). Y enviando (la imagen, guardada previamente en la memoria, al sistema).

El *Código 1* es el código correspondiente a este archivo. Se trata de una memoria, para la cual se ha declarado la señal adicional: rom, un array de WIDTH*HEIGHT posiciones en donde cada posición se encuentra un std_logic_vector de 8 bits, es decir, un píxel de la imagen. Cabe destacar el doble funcionamiento del bloque:

- Si en_guardar='1' entonces la ROM se encuentra guardando los píxeles que manda la cámara, uno a uno, en la memoria. Este guardado se realiza a través de una variable auxiliar (i). Antes de iniciarse el paso de píxeles, esta variable i debe estar a 0, para que el píxel 00 de la imagen se guarde en la posición 0 de la memoria, para ello está el reset_n del bloque. En cada ciclo de reloj, se guarda el píxel contenido por pix_in (el enviado por la cámara) en la posición de memoria i. Además, se incrementa en 1 el valor de i, para que en el siguiente ciclo de reloj el píxel entrante (que se corresponde al píxel contiguo en la imagen) se almacene en la siguiente posición de memoria:

```
if en_guardar='1' then

    if i<=(WIDTH*HEIGHT-1) then
        rom(i)<=pix_in;
        i<=i+1;
    else
        i<=0;
    end if;
```

Además, cuando la i llega a WIDTH*HEIGTH-1, es decir, a la última posición de la memoria, la variable i se reinicia, y a la salida co_guardar del bloque rom.vhd se activa, indicando que ya se ha terminado de guardar la imagen actual:

```
co_guardar<='1' when i=(WIDTH*HEIGHT-1) and en_guardar='1' else '0';
```

De esta forma y a través del bloque de control que se explicará en apartados posteriores, se sincroniza el guardado para que cada imagen se guarde correctamente y no se superpongan informaciones en la memoria.

- Si enable=’1’ el sistema ya tiene la foto guardada en la memoria, y su objetivo es enviarla a través de su salida pix_out. El funcionamiento es simple, en cada ciclo de reloj la ROM pone en la salida el píxel correspondiente a la dirección de memoria addrs. Además, en este modo de funcionamiento no se puede estar guardando una imagen, por lo que se reinicia la variable de control i a 0, para que al guardar una nueva imagen se inicie desde la posición 0.

```
if enable='1' then
    pix_out<=rom(addrs);
    i<=0;
end if;
```

Es importante resaltar que no es compatible que tanto en_guardar como enable estén activas. En este caso, el comportamiento de la rom es un comportamiento indeseado. Es por ello por lo que el bloque de control deberá sincronizar estas dos señales para que no estén nunca activas al mismo tiempo.

4.2.3 LINE_BUFFER.VHD

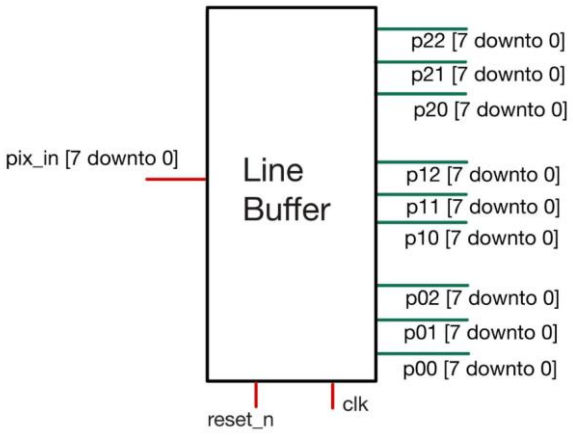


Figura 9. Bloque line_buffer.vhd en VHDL

El propósito de este bloque es disponer los píxeles en ventanas de 3x3 para hacer posible el cálculo del gradiente. Es un bloque fundamental para el diseño posterior del control y temporización del sistema ya que el retardo que supone este bloque es significativo. El código se este bloque se encuentra en el Anexo de códigos: *Código 2.*

El funcionamiento es el siguiente: Se tienen dos arrays (ram1 y ram2) de WIDTH (640) posiciones en donde en cada posición se guardará un píxel (Vivado las infiere como BRAM o como LUTRAM y no como flip-flops, lo cual sería costosísimo). Con el objetivo de no hacer un movimiento excesivo de píxeles, el sistema funciona mediante un puntero (ptr) circular, compartido entre las dos memorias que apunta a una posición de las memorias en cada instante y avanza cada ciclo de reloj.

El registro de entrada, r guarda pix_in un ciclo antes de entrar en las memorias. Se corresponde con la cabeza de la fila actual (fila N). La memoria ram1 guarda r en el lugar donde apunta el putero, donde antes se tenía el valor del píxel de hace WIDTH ciclos. La señal lb1_out registra este valor en cada instante (esto constituye la cabeza de la fila N-1). Por otro lado, ram2(ptr) guarda cada ciclo de reloj lb1_out (el valor del píxel de hace WIDTH ciclos) y lb2_out registra en cada instante el valor de ram2(ptr), es decir, el píxel correspondiente a 2*WIDTH ciclos antes (la cabeza de la fila N-2).

Dentro del proceso síncrono, cada fila tiene dos registros en cadena (_col1, _col0) que simplemente retrasan la "cabeza" 1 y 2 ciclos respectivamente. Así se obtienen las tres columnas de la ventana:

- columna 2 (más reciente): r, lb1_out, lb2_out (las cabezas de fila)
- columna 1 (−1 ciclo): n_col1, m_col1, t_col1
- columna 0 (−2 ciclos): n_col0, m_col0, t_col0

Es importante saber que la ventana correspondiente al píxel[2] entrante aparecerá en la salida WIDTH+2 (642) ciclos después: El camino de un píxel P desde que sale de la rom es el siguiente: P llega al puerto en el ciclo 0, pero solo se captura en r en el siguiente flanco de subida. Así que r = P a partir del ciclo 1. En ese mismo ciclo 1, r se escribe en ram1[ptr]. El puntero ptr recorre las WIDTH posiciones de 0 a WIDTH-1 y vuelve a empezar. Para que lb1_out lea el valor de P, el puntero tiene que completar una vuelta completa y volver al mismo índice. Eso ocurre exactamente WIDTH ciclos después de haberlo escrito: en el ciclo 1 + WIDTH. Gracias a la lectura combinacional (lb1_out <= ram1(ptr)) y no registrada, no hay un ciclo extra. Este valor aparece en la cabeza de la fila N-1: p12. Es un ciclo después cuando P aparece en la posición p11, y en las otras posiciones se encuentran sus vecinos, es decir, en el ciclo WIDTH+2 se tiene la ventana correspondiente a P.

Aunque la primera ventana correspondiente a un píxel de la imagen se dé en el ciclo WIDTH+2, no será hasta el ciclo 2*WIDTH+3 donde se tenga una ventana válida, formada completamente por píxeles de la imagen. Esto se debe a que los píxeles de la primera fila de la imagen no tienen píxeles vecinos como para generar una ventana 3x3 a su alrededor, y las ventanas correspondientes a estos píxeles no serán válidas.

[2] Es decir, cuando el píxel P sea el píxel 11 de la ventana, esa ventana será la correspondiente al píxel P

4.2.4 GRADIENTE.VHD

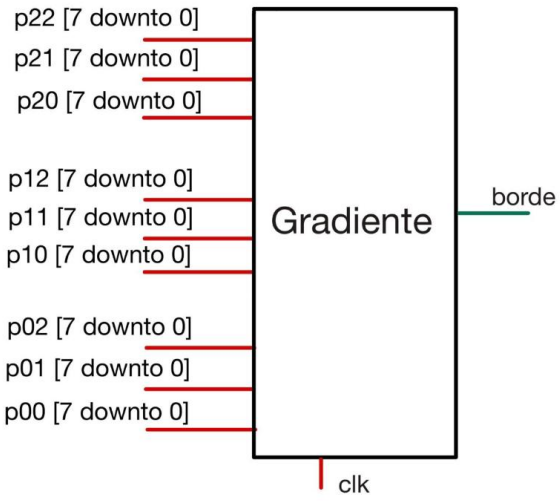


Figura 10. Bloque gradiente.vhd en VHDL

Este bloque recibe una ventana de 3x3 píxeles y, cada ciclo de reloj calcula, a través del establecimiento de un umbral, si el píxel central se considera borde (1) o no borde (0).

Se realiza el cálculo de Gx y Gy multiplicando los elementos de la ventana 3x3 por los elementos de la matriz Kernel: *Figura 2*, explicadas en el Capítulo 1. Este cálculo se actualiza cada ciclo de reloj, por lo que se añade un retraso en la red pipeline. Las señales Gx y Gy son del tipo signed (10 downto 0). Esto se debe a que el resultado puede ser negativo. Además, el valor máximo que pueden alcanzar estas señales son 4*255=1020 (se puede llegar a esta conclusión con una simple observación a las matrices de convolución: *Figura 2*. El caso más desfavorable es tener una profundidad de color de 255 en los pixeles de la última columna (Gx) o primera fila (Gy) de la ventana y una profundidad de 0 en la primera columna o última fila), y el mínimo -1020. Es decir, se necesita una representación en complemento a 2 de 11 bits (el rango de valores en una representación en complemento a 2 con N bits va desde 2^{N-1} hasta $2^{N-1}-1$ [18]). Con 11 bits se podrán representar los números desde el -1024 hasta el 1023.

Para llevar a cabo la operación y que, al guardar el resultado en Gx y Gy, que son señales del tipo signed, no de un error, la suma debe ser de señales de tipo signed de la misma

longitud que el resultado (11 bits). (Si se consulta el código fuente que define la librería numeric_std [22] se observa que la definición de la operación suma para señales tipo signed, devuelve un signed del mismo tamaño que los sumandos). Los píxeles de la ventana son originalmente std_logic_vector de 8 bits. Es necesario hacer un resize de todos los píxeles de la ventana a 11 bits y transformarlo en una señal tipo signed. El tipo de dato std_logic_vector no tiene definida en la librería numeric_std [22] la operación resize, por lo tanto, es necesario transformar los píxeles de la ventana antes de hacer el resize a un dato tipo unsigned:

```
signed(resize(unsigned(p00), 11));
```

Es importante que este primer cambio se haga a unsigned y no directamente a signed, ya que, estas señales representan una profundidad de color, que no puede ser negativa. Si se convirtiese directamente de std_logic_vector a signed, VHDL interpretaría el bit más significativo (MSB) como el bit de signo. Un píxel con valor >= 128 (MSB='1') sería interpretado como un número negativo, y el resize propagaría ese '1' en la extensión.

Los números que tienen que ser multiplicados por 2 (p10 y p12 en Gx, p01 y p21 en Gy) también deben tener la misma longitud que el resto después de haber sido multiplicados. Para esto la mejor opción es hacer la multiplicación con un desplazamiento. Nótese que un desplazamiento de N posiciones a la izquierda de todos los bits significa multiplicar por 2^N y a la derecha dividir por 2^N. De esta forma no se cambia el número de bits del resultado devuelto por la operación:

```
shift_left(signed(resize(unsigned(p10), 11)),1)
```

Así, Gx y Gy se obtienen de la siguiente manera:

```
Gx<=
   -signed(resize(unsigned(p00), 11))
   +signed(resize(unsigned(p02), 11))
   -shift_left(signed(resize(unsigned(p10), 11)),1)
   +shift_left(signed(resize(unsigned(p12), 11)),1)
   -signed(resize(unsigned(p20), 11))
   +signed(resize(unsigned(p22), 11));
```

```
Gy<=
    signed(resize(unsigned(p00), 11))
    -signed(resize(unsigned(p20), 11))
    +shift_left(signed(resize(unsigned(p01), 11)),1)
    -shift_left(signed(resize(unsigned(p21), 11)),1)
    +signed(resize(unsigned(p02), 11))
    -signed(resize(unsigned(p22), 11));
```

Para el cálculo del valor absoluto de G se sigue la aproximación representada en la *Ecuación 2.* Se suman los valores absolutos de Gx y Gy. Como el valor máximo de G es 2*1020=2040, y en este caso G solo puede ser positivo, esta señal se representa con un dato tipo unsigned de 12 bits.

```
G<=resize(unsigned(abs(Gx)),12)+resize(unsigned(abs(Gy)),12);
```

Para decidir finalmente si el píxel es borde o no borde, este valor se compara con un umbral impuesto manualmente:

```
borde<='1' when (G>THESHOLD) else '0';
```

El valor de THRESHOLD se puede cambiar desde el archivo superior de la jerarquía según la precisión necesaria en cada detección. El valor máximo del gradiente es de 2040, pero valores tan grandes en una imagen real son muy poco comunes. Habitualmente los gradientes máximos de una imagen rondan los 900, por lo tanto, si se escoge un umbral superior el mapa de bordes será una imagen en negro (sin bordes). Por el contrario, valores muy pequeños en el umbral detectan bordes excesivamente, y pueden incluir en el mapa de bordes el ruido que tienen algunas imágenes. Valores del umbral entre 50 y 500 son óptimos para la detección.

4.2.5 RAM.VHD

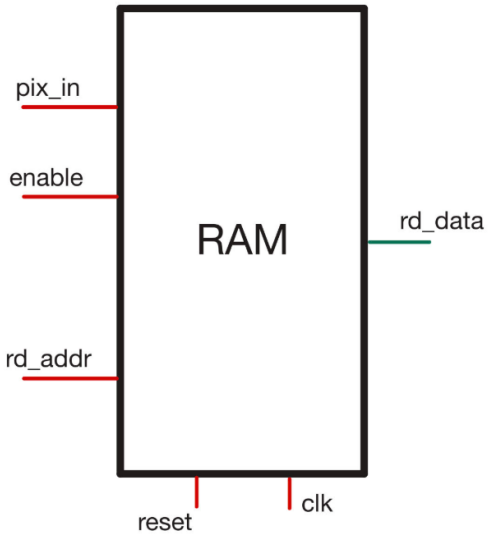


Figura 11. Bloque ram.vhd en VHDL

Este bloque se encarga de almacenar por orden la información que llega por pix_in en los momentos donde enable='1'. Para ello se ha creado una memoria a través de una señal adicional: ram, un array de WIDTH*HEIGHT posiciones, donde en cada posición se almacena un std_logic (esta memoria es más pequeña que la ROM, que almacenaba píxeles en cada posición de memoria. En este caso no se almacenan números de 8 bits, si no de 1 bit).

Se utiliza una variable auxiliar (i). En cada ciclo de reloj se guarda pix_in en ram(i) y se le suma 1 al valor de i, para que el siguiente píxel se almacene en la siguiente posición de memoria:

```
if i<(WIDTH*HEIGHT-1) then
    ram(i)<=pix_in;
    i<=i+1;
else
    i<=0;
end if;
```

Cuando la variable auxiliar llega a la última posición de memoria se reinicia. De todas formas, a partir de este momento el sistema cambiará de estado y no se seguirá guardando información en las primeras posiciones de memoria hasta que la imagen completa sea transmitida a la VGA.

Además, cuenta con un puerto de lectura compuesto por rd_addr y rd_data. El objetivo es que la VGA pueda obtener el mapa de bordes almacenado en la RAM. Para ello se añade un nuevo process, donde la RAM envía por la salida el resultado de borde del píxel que se encuentra en la posición de la memoria rd_addr:

```
process(clk)
    begin
        if rising_edge(clk) then
            rd_data <= ram(rd_addr);
        end if;
    end process;
```

Este process no depende de reset, por lo tanto, aunque se entre en un estado en el que reset de la ram este activo, la VGA podrá leer lo que hay dentro sin problema.

4.2.6 CONTROL.VHD

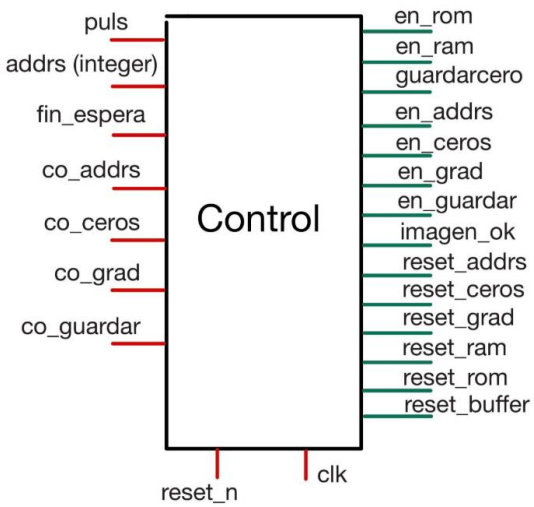


Figura 12. Bloque control.vhd en VHDL

El bloque de control es esencial para la sincronización temporal del sistema. Este bloque activa y desactiva los enables y reset de los contadores, memorias y buffers cuando es necesario. Además, controla la señal guardarcero que se introduce en la puerta and como se ha visto en la *Figura 7. Estructura general simplificada de Sobel en VHDL.* Esta señal se pone a 0 en los momentos en los que, la salida del bloque gradiente se corresponde con un píxel esquina, ya que para estos píxeles el cálculo del gradiente, y por ende la determinación de borde o no borde, no son válidas. Este bloque controla también la señal imagen_ok, la señal que se le enviará a la cámara para indicarle que debe capturar una nueva imagen.

El bloque está diseñado como una máquina de estados. El diagrama de estados se presenta en la *Figura 13*.

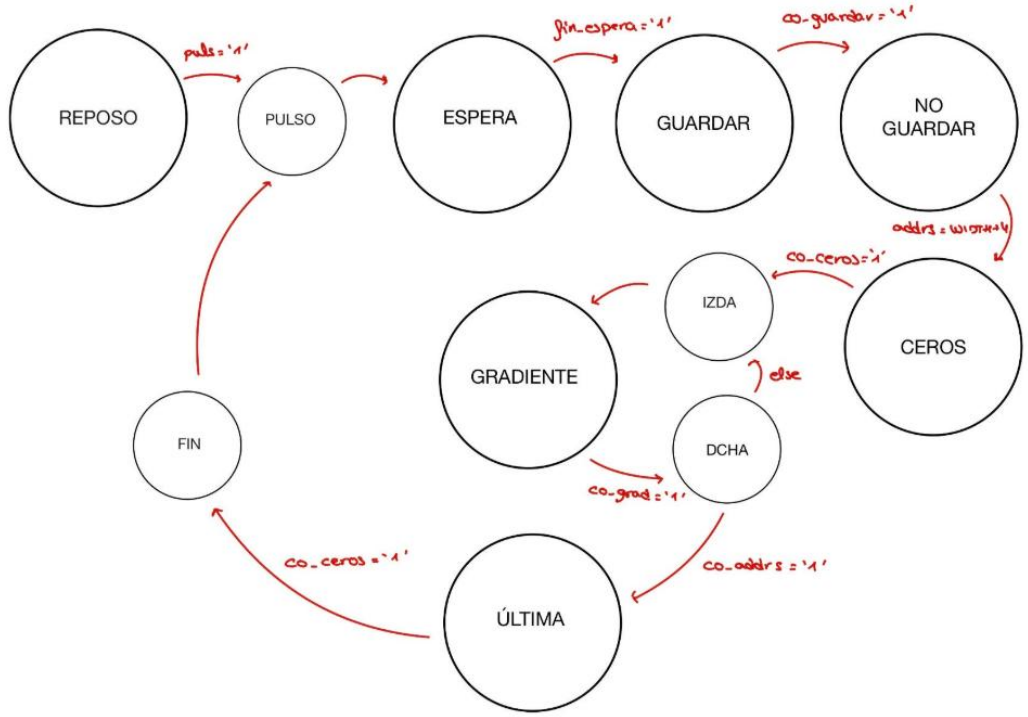


Figura 13. Diagrama de estados del bloque de control en Sobel

Se parte de un estado de reposo. Cuando se pulse un pulsador (puls) se inicia el sistema y se pasa a un estado transitorio, Pulso, que activa la señal imagen_ok=’1’ para indicarle a la

cámara que debe capturar una nueva imagen. De este estado se pasa directamente a un estado de espera, pues solo es necesario tener imagen_ok activa durante un ciclo de reloj:

```
when Pulso =>
      estado_sig<=Espera;
```

La cámara comienza a capturar la imagen, y el sistema entra en un estado de Espera. Cuando la cámara está lista para empezar a enviar los píxeles, fin_espera se pone a '1', y habrá que cambiar de estado:

```
when Espera =>
    if fin_espera='1' then
        estado_sig<=Guardar;
    end if;
```

En estos tres últimos estados (Reposo, Pulso y Espera) el sistema simplemente se está comunicando con el exterior. Los reset de los componentes deben estar todos a 1, y los enable a 0, no hay nada funcionando por el momento, por lo que es bueno reiniciar las variables de cada bloque.

En el estado Guardar se va rellenando la memoria hasta llegar al último píxel de la imagen, donde la ROM envía una señal (co_guardar='1'), se debe cambiar de estado:

```
when Guardar =>
    if co_guardar='1' then
        estado_sig<=NoGuardar;
    end if;
```

En el estado Guardar, únicamente se desactiva el reset de la rom y activa en_guardar, el resto de los componentes siguen sin tener una función en este momento.

Sin embargo, en el estado en los siguientes estados ya se tiene la imagen completa en la ROM, y se procede con el pipeline. La ROM entra en el modo de envío, por lo tanto, en_guardar='0' y en_rom='1'. Los píxeles tienen que ir saliendo uno a uno. Para ello se

activa el cont_addrs (en_addrs='1' y reset_addrs='1'), que proporciona la señal addrs, que cuenta de 0 a WIDTH*HEIGHT-1, recorriendo así todas las posiciones de la imagen. El linebuffer también entra en funcionamiento, por lo que el reset_buffer se tiene que desactivar. Además, el estado NoGuardar es un estado en el que, el resultado obtenido en la salida del bloque gradiente.vhd todavía no es el referente a ninguno de los píxeles de la imagen, los primeros píxeles todavía están entrando al sistema, y no han llegado al final, por lo tanto, este resultado no se quiere guardar en la RAM: en_ram='0'. Recuérdese que el primer píxel entra en el sistema y cuenta con ciertos retardos: retardo para salir de la ROM (la rom saca cada ciclo de reloj un dato), WIDTH+2 retardos en el linebuffer y otro retardo en el cálculo del gradiente. Así, el resultado del píxel 00 de la imagen se verá reflejado en la salida del bloque gradiente en el ciclo WIDTH+4, es decir, cuando addrs=WIDTH+4. Es en este momento cuando se debe cambiar de estado para comenzar a guardar resultados en la ram:

```
when NoGuardar =>
                if addrs>=(WIDTH+4) then
                    estado_sig<=Ceros;
                end if;
```

En el estado Ceros ya se guardan los resultados (en_ram='1', reset_ram='0'), pero al ser píxeles esquina, el resultado que se guarda debe ser un '0' (guardarcero='0'). Durante la primera fila esto no cambia, por lo que se activa un contador (en_ceros='1', reset_ceros='0') que cuenta hasta WIDTH-1. De esta forma, cuando co_ceros='1' se habrá llegado a la segunda fila y habrá que cambiar de estado:

```
when Ceros =>
    if co_ceros='1' then
        estado_sig<=Izda;
    end if;
```

En las filas intermedias de la imagen, únicamente los píxeles de los extremos (primero y último de cada fila) deben almacenarse como 0 en la RAM. Para gestionar esto, se emplea un contador cont_grad que se activa en el estado Gradiente y contabiliza los píxeles interiores

válidos hasta alcanzar WIDTH-3, cuando se activa co_grad='1', por lo que se pasa al estado Dcha, el sistema se encuentra procesando el último píxel válido de la fila (guardarceros='0'):

```
when Gradiente=>
                if co_grad='1' then
                    estado_sig<=Dcha;
                end if;
```

Desde este estado se evalúa si la fila en curso es la penúltima de la imagen: Cuando el último valor de la imagen esta saliendo de la ROM (addrs=WIDTH*HEIGHT-1 →co_addrs='1')[3] en la salida del bloque gradiente se procesa el píxel de WIDTH+4 ciclos antes, esto quiere decir que el sistema se encuentra en la penúltima fila. En este caso se pasará a un estado: Ultima, donde las condiciones son las mismas que en el estado Ceros. En caso contrario se pasa al estado Izda, con las mismas condiciones que Dcha:

```
when Dcha=>
                if co_addrs='1' then
                    estado_sig<=Ultima;
                else
                    estado_sig<=Izda;
                end if;
```

```
when Izda=>
                estado_sig<=Gradiente;
```

De esta forma el ciclo Gradiente →Dcha →Izda →Gradiente se repite hasta alcanzar la última fila. En los estados Izda y Dcha se reinicia cont_addrs.

Al finalizar la última fila (co_ceros='1') se pasa a un estado Fin, de un ciclo de reloj de duración y que reinicia todos los componentes. De este estado se pasa directamente a Pulso, para pedir otra imagen a la cámara y que el sistema esté funcionando en bucle.

[3] Se ha diseñado cont_addrs para que cuente hasta WIDTH*HEIGHT-1 y no se reinicie de no ser porque se le aplica un reset. Cuando llega a ese valor mantiene co_addrs='1'.

4.2.7 CONT_ADDRS.VHD

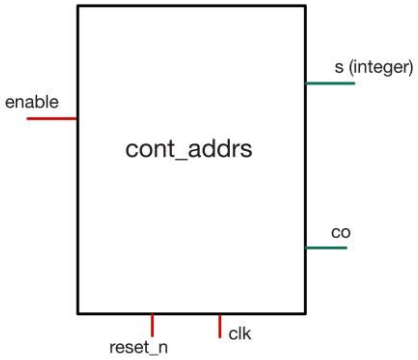


Figura 14. Bloque cont_addrs.vhd en VHDL

Se trata de un contador de módulo WIDTH*HEIGHT-1 con una única peculiaridad, cuando acaba la cuenta no se reinicia, y mantiene el valor del rebose a 1. El código correspondiente es el *Código 7.*

4.2.8 CONT_CEROS.VHD

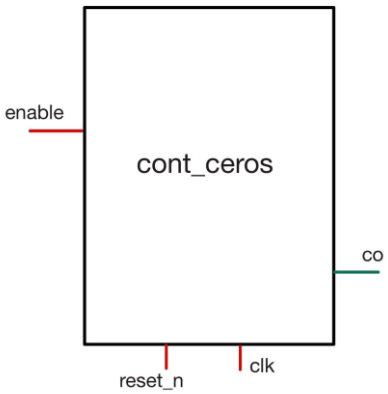


Figura 15. Bloque cont_ceros.vhd en VHDL

Es un contador que cuenta hasta WIDTH-1 con habilitación de cuenta y rebose [18]. El código correspondiente es *Código 8*

4.2.9 CONT_GRAD.VHD

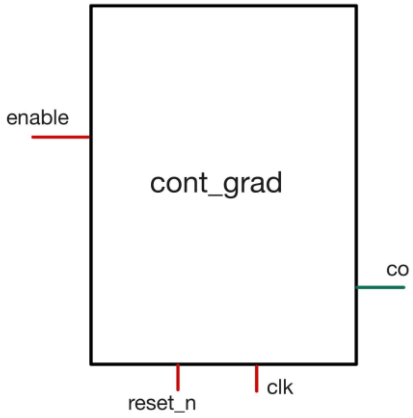


Figura 16. Bloque cont_grad.vhd en VHDL

Es un contador que cuenta hasta WIDTH-3 con habilitación de cuenta y rebose [18]. El código correspondiente es Código 9.

4.2.10 SOBEL.VHD

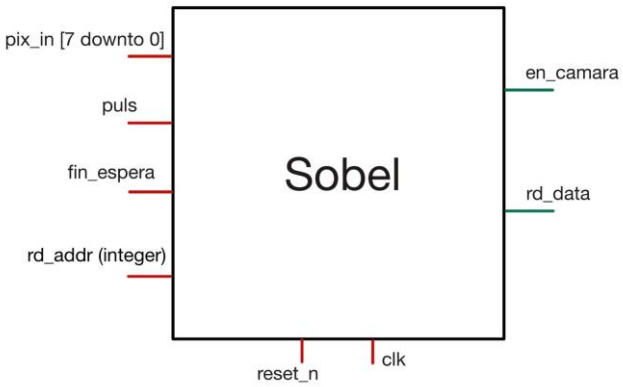


Figura 17. Bloque sobel.vhd en VHDL

Este bloque es el bloque mas alto de la jerarquía. Es un bloque estructural que establece las conexiones necesarias entre el resto de los bloques. El diagrama de conexión final es el que se observa en la *Figura 18*. Tiene como entradas y salidas las señales necesarias para la

conexión del sistema con el exterior (pulsador, cámara y VGA). El *Código 10* es el correspondiente a este bloque.

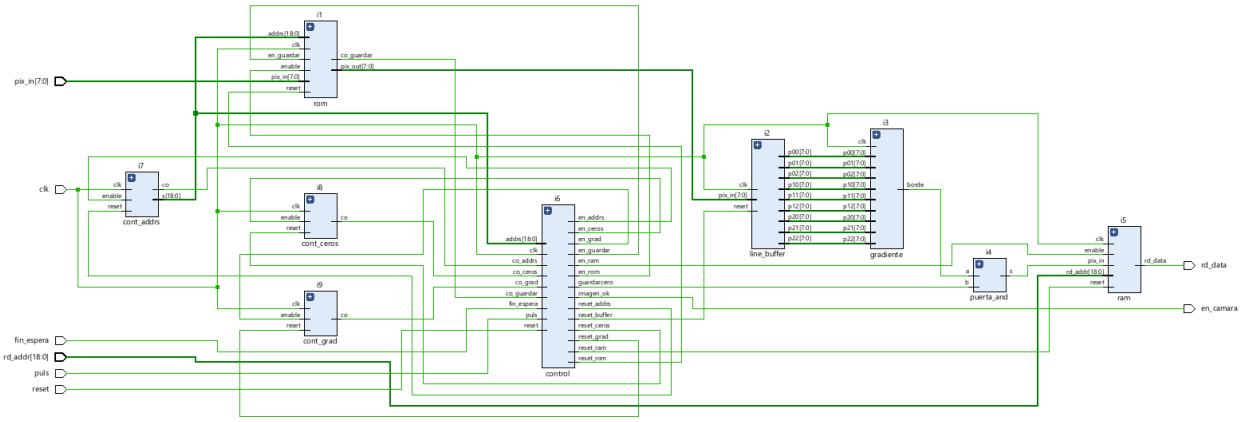


Figura 18. Estructura general de Sobel en VHDL

4.2.11 SIMULACIÓN

Para poder llevar a cabo una simulación controlada se ha realizado con una imagen 10x10, que tiene la columna 3 llena de píxeles negros (intensidad=0 (x00)) y el resto de la imagen se compone de pixeles blancos (intensidad=255 (xff)). Para ello se han cambiado los generics del proyecto a WIDTH=10 y HEIGHT=10. Además, se ha diseñado un testbench que interactúe con el sistema de detección de bordes Sobel. El código de testbench se refleja en: *Código 31*. Este testbench proporciona la señal de reloj, que sincroniza el sistema:

```
clk_process : process
    begin
        clk_tb <= '0';
        wait for clk_period/2;
        clk_tb <= '1';
        wait for clk_period/2;
    end process;
```

Además, el testbench replica el comportamiento del pulsador que inicia el sistema y de la cámara: espera a recibir un pulso (en_camara=’1’) para comenzar a “capturar” una imagen.

Se esperan 45ns, simulando el tiempo de captura de la cámara y se envía un pulso al sistema (fin_espera=’1’):

```
wait until rising_edge(clk_tb) and en_camara_tb='1';
        wait for 45 ns;         --tiempo que tarda la camara en capturar la imagen
        fin_espera_tb<='1';
        wait for clk_period;
        fin_espera_tb<='0';
```

Después de esto se comienza con el envío de los píxeles formando la imagen descrita anteriormente (columna 3 de píxeles negros):

```
for fila in 0 to HEIGHT-1 loop
            for columna in 0 to WIDTH-1 loop
                ------------------------------------------------------
                -- Columna negra
                ------------------------------------------------------
                if columna = 3 then
                    pix_in_tb <= x"00";

                ------------------------------------------------------
                -- Resto blanco
                ------------------------------------------------------
                else
                    pix_in_tb <= x"FF";

                end if;

                wait until rising_edge(clk_tb);
            end loop;
        end loop;
```

El resultado de la simulación es el siguiente:

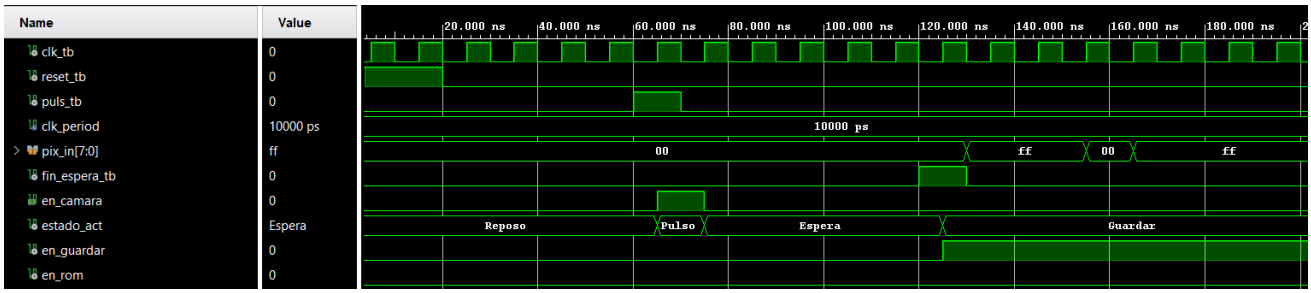


Figura 19. Estados Reposo, Pulso, Espera y Guardar de la simulación sobel.vhd

El sistema inicia en el estado reposo, y cuando recibe un pulso pasa al estado Pulso, donde, como se ve en la *Figura 19*, la salida del bloque sobel en_camara se activa, indicándole a la cámara que debe capturar una imagen. Despues de esto se entra en el estado Espera, y cuando fin_espera=’1’ se pasa al estado Guardar. El testbench, tal y como se ha programado, comienza a enviar los píxeles de la imagen diseñada con una línea en la columna 3. Cuando finaliza el guardado, co_guardar=’1’ y se pasa al siguiente estado: NoGuardar (esto se aprecia ya en la siguiente imagen: *Figura 20*).

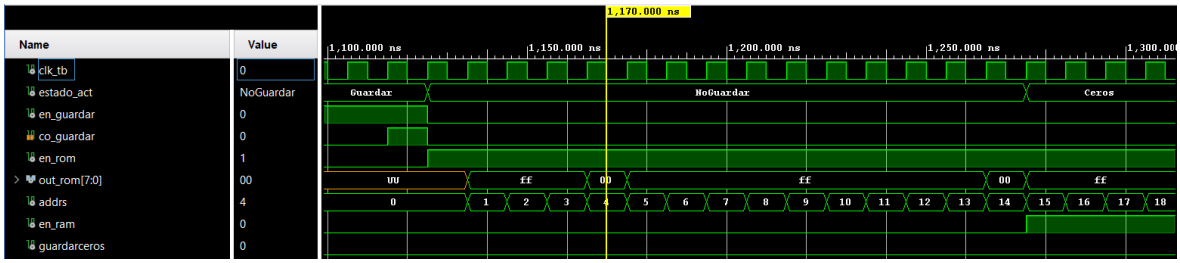


Figura 20. Estados NoGuardar y Ceros de la simulación sobel.vhd

En el estado NoGuardar se habilita la ROM (en_rom=’1’), pero la RAM todavía no (en_ram=’0’), como estaba previsto. Además, se inicia el cont_addrs y comienza a salir la imagen guardada por la ROM (out_rom). Cuando addrs=WIDTH+4 (14 en el caso de la simulación) el sistema cambia de estado, tal y como se había definido.

En el estado Ceros se habilita la RAM (en_ram=’1’), pero todavía se guardan ceros (guardarceros=’0’). Cuando co_ceros=’1’ se pasa a la segunda fila, y se comienza el bucle Gradiente→Dcha→Izda→Gradiente. En la *Figura 21* se puede ver el desarrollo de este bucle. Además, se ha dispuesto la señal auxiliar i de la ram, para ver en cada instante la

posición de la memoria donde se guarda in_ram. Así se ve claramente que se guardan píxeles como bordes alrededor de la columna 3. Se observa también el correcto funcionamiento de las esquinas, cuando el sistema entra en los estados Izda y Dcha, guardarceros=’0’.

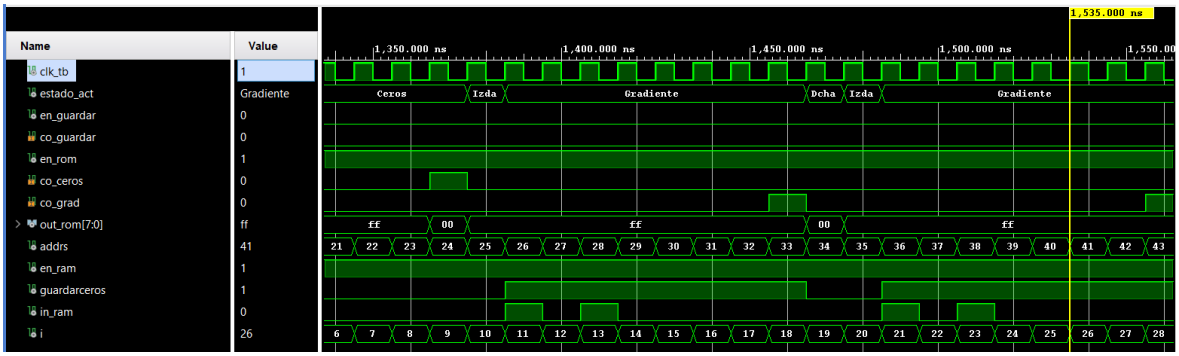


Figura 21. Estados Gradiente, Izda y Dcha en la simulación sobel.vhd

Cuando co_addrs=’1’ y se llegue a un píxel esquina derecha, se deberá de cambiar de estado a Ultima. El correcto funcionamiento en este aspecto se ve en la *Figura 22.* En este estado, guardarceros=’0’. Cuando co_ceros se ha acabado de guardar la imagen, y el sistema pasa al estado Fin, donde se resetean las variables para volver a empezar, enviando el pulso a la cámara.

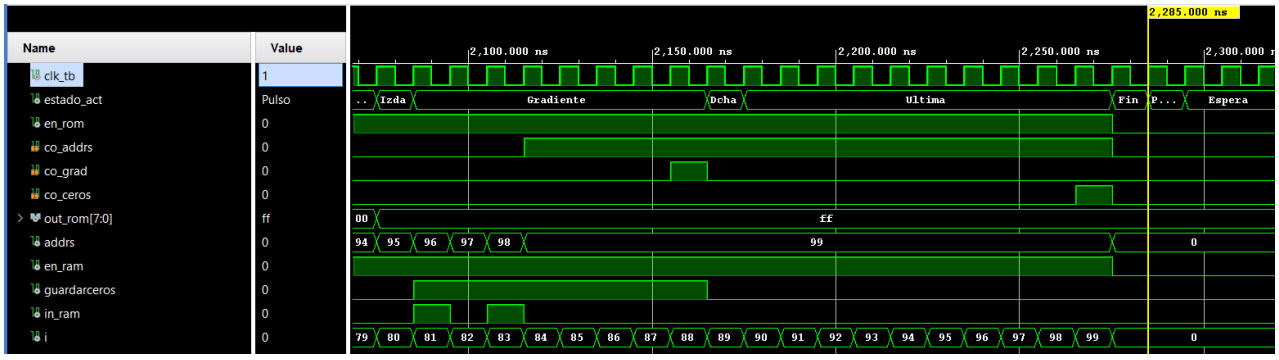


Figura 22. Estados Ultima y Fin en la simulación sobel.vhd

A partir de esta simulación se puede deducir cuanto tardará el sistema en realizar la detección de bordes completa: La detección acaba en el instante donde el tiempo de simulación es de 2.285ns (*Figura 22*) y empieza en el 1.125ns (*Figura 20*) (no se contempla el tiempo de

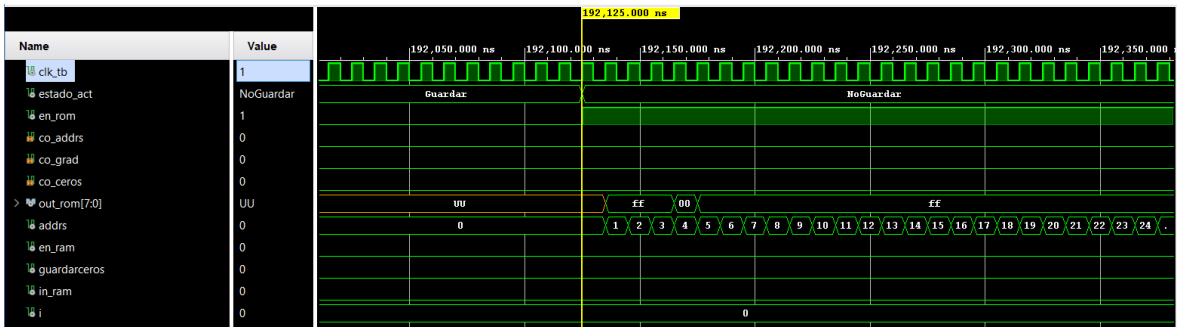
guardado, por lo que se considera que la detección comienza cuando acaba el estado Guardar)

$$2.285 - 1.125 = 1.160ns\ de\ simulación$$

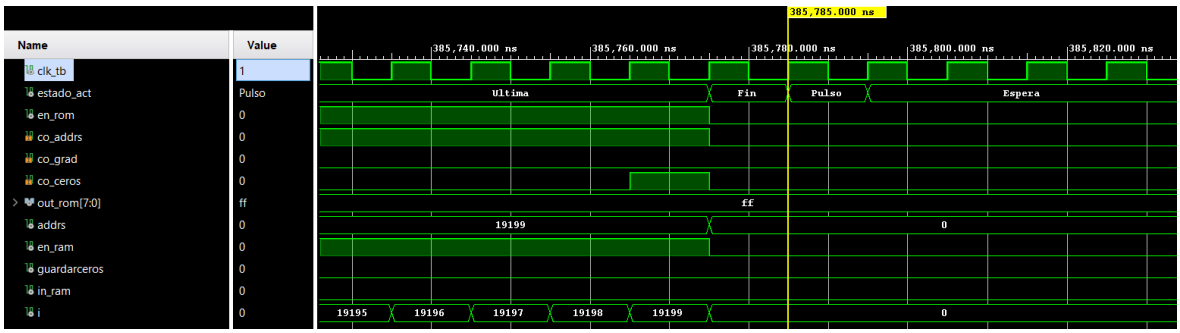
$$\frac{2.160ns}{Tclk\ (sim)} = \frac{2.160ns}{10ns} = 116\ ciclos\ de\ reloj$$

La placa Basys3 incluye un oscilador de 100 MHz conectado al pin W5 [23], por lo tanto, se tardarán 1.160ns en la detección de bordes de una imagen 10x10.

Si hacemos la prueba para una imagen 160x120: La detección comienza en el 192.125ns



Y finaliza en el 385.785ns



Así el sistema tarda 19.366 ciclos de reloj, es decir, con el reloj de la FPGA: 193.660ns

Para una imagen 320x240: La detección comienza en el 768.125ns

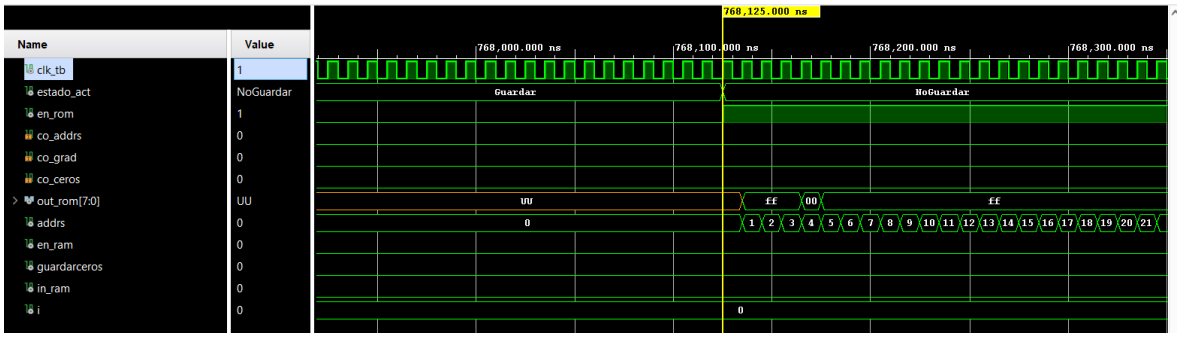


Figura 23. Inicio simulación imagen 320x240 de sobel.vhd

Y finaliza en el 1.539.385ns

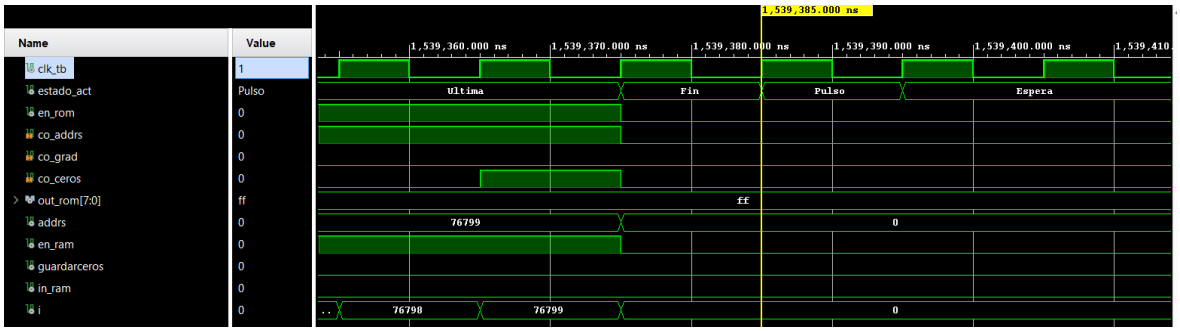


Figura 24. Final simulación imagen 320x240 de sobel.vhd

Así el sistema tarda 77.126 ciclos de reloj, es decir 771.260ns.

Con una imagen 640x480: La detección comienza en el 3.072.125ns

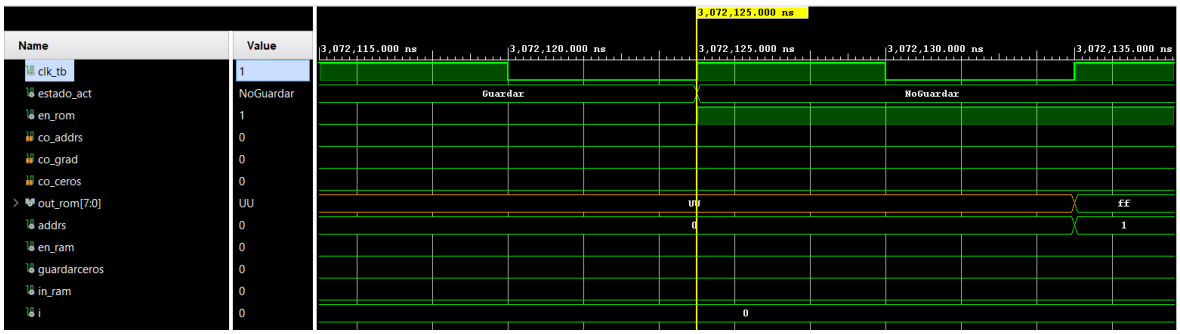


Figura 25. Inicio simulación imagen 640x480 de sobel.vhd

Y finaliza en el 6.150.585ns

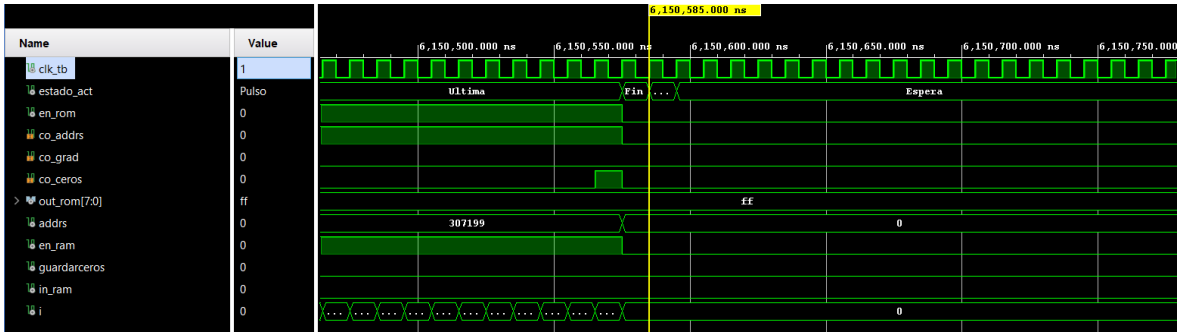


Figura 26. Final simulación imagen 640x480 de sobel.vhd

Se tardará 3.078.460ns en completar la detección.

4.3 DISEÑO CANNY

En este apartado se detallará el desarrollo en VHDL del algoritmo de detección de bordes Canny, así como su simulación en el tiempo mediante la herramienta Vivado 2025.2.

4.3.1 ESTRUCTURA GENERAL

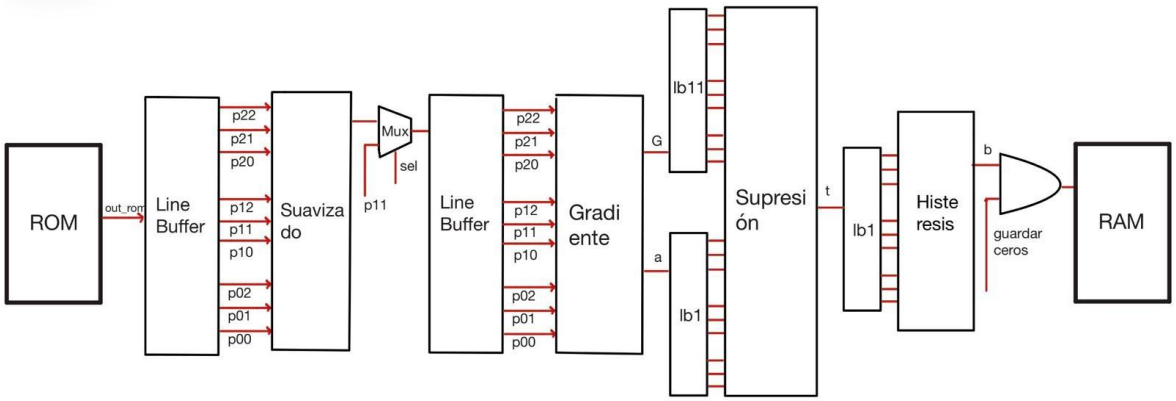


Figura 27. Estructura general simplificada de Canny en VHDL

La estructura es muy parecida a la del algoritmo Sobel, añadiendo las etapas extra que necesita el algoritmo Canny para su correcto funcionamiento (Suavizado, Gradiente, Supresión de no máximos (en este bloque se establece también el doble umbral, por lo que la salida del bloque representa el tipo de borde (débil-fuerte-no borde)) e Histéresis. En este caso, son dos las señales controladas por el bloque de control. Una de ellas ya es conocida, guardarceros. La otra es sel, la señal de selección de entrada en el multiplexor. Esto se debe a que ahora se tiene que controlar también cuando la salida del bloque suavizado se corresponde con un píxel esquina. Cuando esto sucede, el valor de profundidad de color utilizado para el cálculo de gradiente es el valor original del píxel (sel='0') ya que el proporcionado por el suavizado es erróneo. De esta forma el proceso del cálculo de gradiente y determinación de los píxeles como bordes o no bordes se hace sobre una nueva imagen, donde todos los píxeles han pasado por un filtro de suavizado gaussiano, a excepción de los píxeles esquina, que mantienen su profundidad de color original.

Igual que antes, se muestra una estructura simplificada de Canny para un mejor entendimiento inicial: *Figura 27.* La estructura completa se muestra en la *Figura 36.*

4.3.2 ROM.VHD

Este bloque es idéntico al desarrollado para el algoritmo Sobel. Para comprender su funcionamiento véase *4.2.2 rom.vhd.*

4.3.3 LINE_BUFFER.VHD

Este bloque es idéntico al desarrollado para el algoritmo Sobel. Para comprender su funcionamiento véase *4.2.3 line_buffer.vhd.*

4.3.4 SUAVIZADO.VHD

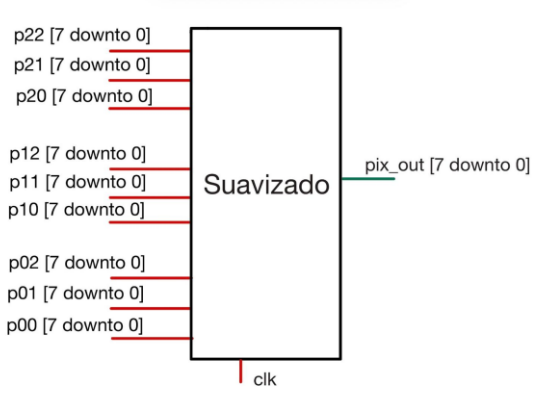


Figura 28. Bloque suavizado.vhd en VHDL

Este bloque tiene el objetivo de aplicar el suavizado gaussiano explicado en la *sección 1.2.2.* Para ello se multiplica cada elemento de la matriz 3x3 de píxeles por cada elemento de la matriz de referencia del suavizado: *Figura 3*. Para realizar las multiplicaciones y divisiones se realizan desplazamientos. El resultado es un número del mismo tamaño que el original (8 bits) ya que, si se observa la matriz de convolución, el resultado está normalizado (al estar dividido por 16). Como el resultado es un número del mismo tamaño que los píxeles iniciales, no hace falta hacer un resize. De esta forma, el cálculo de la nueva intensidad de color del píxel central:

```
intensidad<=shift_right(unsigned(p00),4)
                        +shift_right(unsigned(p01),3)
                        +shift_right(unsigned(p02),4)                -- 2/16=1/8
                        +shift_right(unsigned(p10),3)
                        +shift_right(unsigned(p11),2)
                        +shift_right(unsigned(p12),3)                -- 4/16=1/4
                        +shift_right(unsigned(p20),4)
                        +shift_right(unsigned(p21),3)
                        +shift_right(unsigned(p22),4);
```

Este cálculo se actualiza cada ciclo de reloj, por lo que introduce un retraso en la red pipeline.

4.3.5 GRADIENTE.VHD

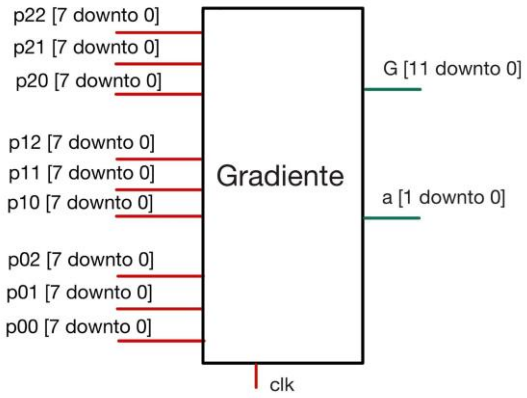


Figura 29. Bloque gradiente.vhd en VHDL

Este bloque calcula el gradiente de la ventana 3x3 entrante y, a diferencia del bloque gradiente.vhd perteneciente al proyecto Sobel, también calcula el ángulo del gradiente (a[1 downto 0]). El cálculo del gradiente es idéntico al explicado en la *sección 4.2.4 gradiente.vhd.* Este cálculo se actualiza cada ciclo de reloj, por lo que introduce un retraso en la red pipeline.

El ángulo se aproxima a uno de estos 4 valores: 0º, 45º, 90º y 135º. Esto es así, porque solo en estas cuatro direcciones existen píxeles vecinos (para la posterior comparación de gradientes en la etapa de supresión de no máximos). Para esta aproximación:

```
angulo<= "00" when (abs_gx>shift_left(abs_gy,1)) else
            "10" when (abs_gy>shift_left(abs_gx,1)) else
```

```
            "01" when (Gx>0 and Gy>0) or (Gx<0 and Gy<0) else
            "11";
```

Se representa el ángulo con un número de 2 bits:

- 00→0º
- 01→45º
- 10→90º
- 11→135º

El código completo de este bloque es el *Código 15*.

4.3.6 LINE_BUFFER11.VHD Y LINE_BUFFER1.VHD

Estos bloques tienen el mismo funcionamiento que el *line_buffer.vhd*, la única diferencia es que componen ventanas 3x3 de std_logic_vector de 12 y 2 bits, en lugar de 8. La única diferencia en el código es el cambio en los parámetros de tamaño de las señales. Los códigos correspondientes son: *Código 16* y *Código 17* respectivamente.

4.3.7 SUPRESION.VHD

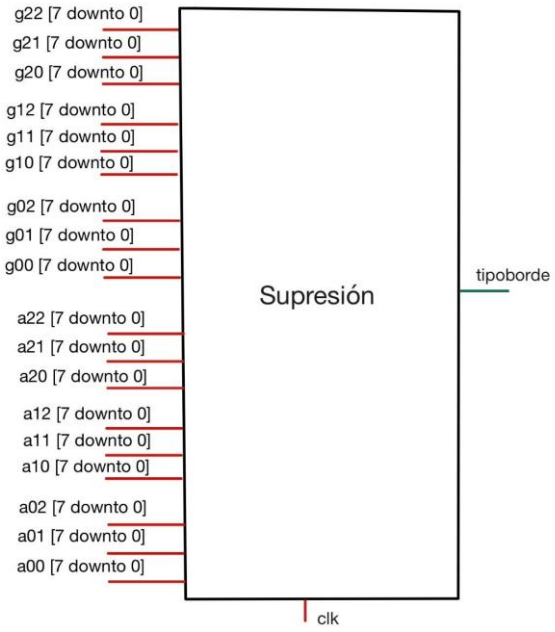


Figura 30. Bloque supresion.vhd en VHDL

Este bloque engloba la etapa de supresión de no máximos del algoritmo Canny ("adelgazar" los bordes para que sean bordes de 1 píxel de grosor a través de comparar si es el máximo local) y la de establecimiento de doble umbral (se establece el tipo de borde, para en etapas posteriores realizar la histéresis). Para ello es necesario la ventana 3x3 de gradientes de los píxeles y la de direcciones (en realidad solo se necesita la dirección del píxel central, ya que es el píxel de estudio, pero la temporización se vuelve mas sencilla utilizando un buffer también para las direcciones, así se sabe que, en cada instante, la dirección correspondiente con los gradientes de la ventana son los correspondiente en la ventana de direcciones).

El procedimiento es el siguiente:

- Si el píxel central (el de estudio) no es un máximo local, directamente se considera como no borde. Para comprobar esto se analiza la dirección del gradiente central, y se compara la magnitud con los dos píxeles vecinos en esta dirección. Por lo tanto, según la dirección (a11) se deberá comparar con unos píxeles u otros:

- a11=00 → El ángulo es 0º y se deberá comparar el píxel central con los píxeles 10 y 12 de la ventana
- a11=01 → El ángulo es 45º y se deberá comparar el píxel central con los píxeles 02 y 20 de la ventana
- a11=10 → El ángulo es 90º y se deberá comparar el píxel central con los píxeles 01 y 21 de la ventana
- a11=11 → El ángulo es 135º y se deberá comparar el píxel central con los píxeles 00 y 22 de la ventana

El píxel se considera no borde también si el gradiente es inferior al umbral bajo[4]. Así, la condición para considerar a un píxel no borde:

```
if unsigned(g12)>unsigned(g11) or unsigned(g10)>unsigned(g11) or
   unsigned(g11)<Tbajo then
         tipoborde<="00";
```

Se muestra el ejemplo para a11=00, para el resto de los casos basta con cambiar los píxeles vecinos con los que se compara g11. Se pasa el valor de los gradientes a unsigned para poder hacer la comparación, ya que para std_logic_vector no está definida la operación > o < [22].

- Si no se cumple ninguno de esos supuestos (g11<Tbajo ni g11<pixeles vecinos) se pasará a comprobar si el valor de g11 es mayor o menor que Talto, para definirlo como borde fuerte o débil:

```
elsif unsigned(g11)>Talto then
    tipoborde<="10";               --borde fuerte
else
    tipoborde<="01";               --borde débil
end if;
```

Este cálculo se actualiza cada ciclo de reloj, por lo que introduce un retraso en la red pipeline.

[4] Los umbrales, alto y bajo se establecen manualmente en el archivo de VHDL y se pueden cambiar para modular la precisión de detección.

4.3.8 HISTERESIS.VHD

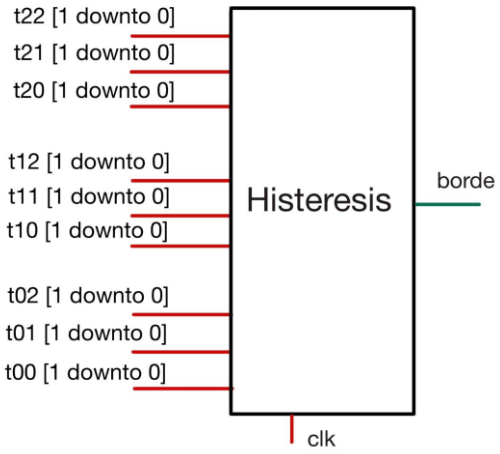


Figura 31. Bloque histeresis.vhd en VHDL

El último paso es la eliminación de los bordes débiles independientes, que no están conectados a ningún bloque fuerte (*sección 1.2.2*). El bloque histeresis.vhd recibe una ventana 3x3 de tipos de borde (débil fuerte o no borde). Se analiza el píxel central (t11). La regla es:

- Si el borde central es no borde originalmente, será no borde de la imagen:

```
if t11="00" then
    borde<='0';
```

- Si el borde central es borde fuerte originalmente, será borde de la imagen:

```
elsif t11="10" then
    borde<='1';
```

- Si el borde central es borde débil originalmente, se comparará con los bordes vecinos. Si alguno de ellos es fuerte, el borde se mantiene, de lo contrario, el píxel no será un borde de la imagen:

```
elsif t11="01" then
    if t00="10" or t01="10" or t02="10" or t10="10"
```

```
    or t12="10" or t20="10" or t21="10" or t22="10" then
        borde<='1';
    else
        borde<='0';
    end if;
```

Este cálculo se actualiza cada ciclo de reloj, por lo que introduce un retraso en la red pipeline.

4.3.9 RAM.VHD

Este bloque es idéntico al desarrollado para el algoritmo Sobel. Para comprender su funcionamiento véase *4.2.5 ram.vhd.*

4.3.10 CONTROL.VHD

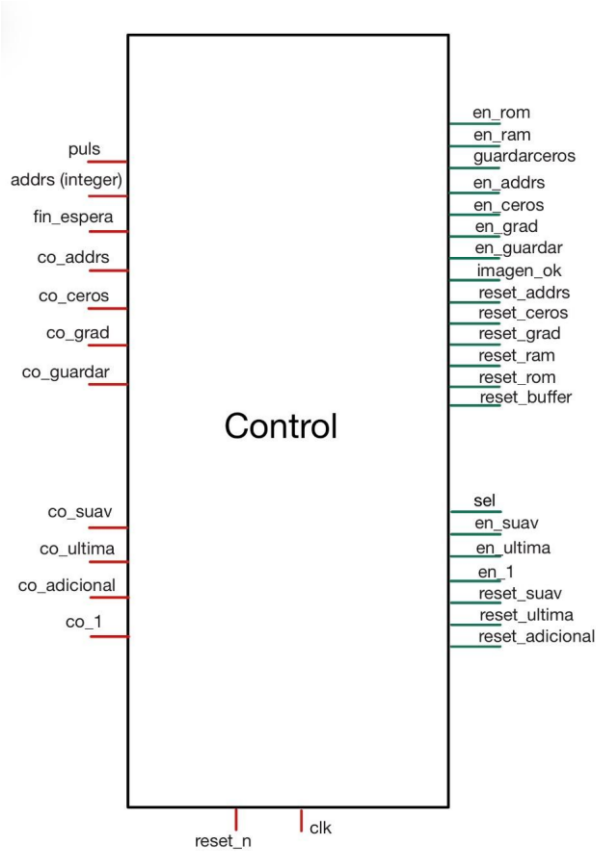


Figura 32. Bloque control.vhd en VHDL

Este bloque es muy similar al bloque *4.2.6 control.vhd* del proyecto Sobel. También se trata de una máquina de estados donde el objetivo es que, en los momentos donde en la salida del bloque suavizado o en la salida del bloque histéresis se encuentre el resultado correspondiente a un píxel esquina, el sistema escoja la profundidad de color original del píxel (sel='0') o guarde un 0 en la RAM (guardarceros='0') respectivamente. El código que se corresponde con este bloque es el *Código 22*.

En este caso es necesario incluir más estados en el sistema, ya que, es posible que, en el momento que un píxel esquina esté en la salida del suavizado, en la salida de histéresis se tenga un píxel central (sel='0' pero guardarceros='1') o viceversa. Para ello se han creado los estados dchaS, izdaS (se activan cuando hay píxeles esquina saliendo por el bloque suavizado. sel='0' y guardarceros='1'), dchaG, izdaG (se activan cuando hay píxeles esquina saliendo por el bloque histéresis. sel='1' y guardarceros='0'). También es posible que por ambos esté saliendo el resultado de un píxel esquina. Para ello se han creado los estados dchaSG, izdaSG, dchaSizdaG, izdaSdchaG (sel='0' y guardarceros='0').

El mecanismo es el mismo que el diseñado para Sobel. En este caso se necesitan separar los contadores que controlan la salida del suavizado con los que controlan la salida de la histéresis. Así, se tienen los mismos que antes para el control de la salida del bloque histéresis: cont_ceros (para contar la primera fila y la última) y cont_grad (para contar las filas intermedias). Estos contadores se reinician únicamente en los estados donde hay un píxel esquina en la salida del bloque histéresis (dchaG, izdaG, dchaSG, izdaSG, dchaSizdaG y izdaSdchaG). Además, se tienen los contadores correspondientes para el control de la salida del suavizado: cont_ultima (para contar la última fila) y cont_suav (para contar las filas intermedias). Estos contadores se reinician únicamente en los estados donde hay un píxel esquina en la salida del bloque suavizado (dchaS, izdaS, dchaSG, izdaSG, dchaSizdaG y izdaSdchaG).

Las etapas iniciales (Reposo, Pulso, Espera y Guardar) de comunicación con la cámara son idénticas a las del bloque del bloque *control.vhd* del proyecto Sobel. A continuación, se entra en un estado NGyP, en el cual, todavía no se guardan los resultados en la RAM (en_ram='0')

y el valor seleccionado para la entrada del gradiente es la profundidad original de color (sel='0'), ya que los resultados obtenidos en la salida del bloque histeresis.vhd y suavizado.vhd todavía no son los referentes a ninguno de los píxeles de la imagen, los primeros píxeles todavía están entrando al sistema, y no han llegado al final. No es hasta que en la salida del bloque suavizado.vhd se tiene el resultado correspondiente al píxel 11 (el primero de los píxeles centrales de la imagen en llegar) cuando se cambia de estado. Esto ocurre 2*WIDTH+5 ciclos después (p11 sale de la ROM en el ciclo WIDTH+2, el linebuffer aplica WIDTH+2 retrasos, y el bloque suavizado otro retraso):

```
when NGyP =>
    if addrs=(2*WIDTH+5) then
        estado_sig<=NGyS;
    end if;
```

Se pasa a una etapa en la que, en la salida del bloque suavizado.vhd ya se están procesando filas centrales, mientras que en la salida del bloque histeresis.vhd todavía no se tiene el resultado correspondiente a ningún píxel de la imagen (en_ram='0'). En esta etapa se inicia el contador correspondiente al suavizado (cont_suav) y se activan las etapas dchaS y izdaS cuando sean necesarias. Así, se tiene un funcionamiento similar al de *control.vhd* de Sobel, teniendo un bucle NGyS→dchaS→izdaG→NGyS. Este bucle se repite hasta que en la salida del bloque histeresis.vhd se tiene el resultado correspondiente al píxel 00 de la imagen (esto ocurre 4*WIDTH+13 ciclos después ya que se introducen 4*WIDTH+8 retrasos en los 4 linebuffers, un retraso de salida en la ROM, un retraso en el bloque suavizado.vhd, un retraso en el bloque gradiente.vhd, uno en el bloque supresion.vhd y otro en el bloque histeresis.vhd), donde cambian las condiciones y es necesario cambiar de estado para comenzar a guardar los resultados (en_ram='0'):

```
 when NGyS =>
                if co_suav='1' then
                    estado_sig<=dchaS;   --en dchaS el en_ceros esta activo, por
lo que si coincide que justo en addrs=(4*WIDTH+12) (que es cuando en la salida
del bloque histéresis sale el valor correspondiente al píxel p00), con que
co_suav='1', en dchaS ya estaremos activando el contador para contar la primera
fila.
                elsif addrs=(4*WIDTH+13) then
```

```
                    estado_sig<=CyS;
                end if;
```

Se entra en una etapa donde en la salida del bloque histeresis.vhd se encuentra el resultado de un píxel de la primera fila, y en la salida del bloque suavizado se está procesando una de las filas intermedias. Así, se activa el contador cont_ceros (en_ceros='1' y reset_ceros='0') y se mantiene en funcionamiento el bucle controlado por cont_suav: CyS→dchaS→izdaS→CyS, hasta que co_ceros='1'.

En este momento, el sistema se encuentra procesando filas intermedias en ambas salidas. Por lo tanto, se activan los contadores cont_suav y cont_grad, para activar los estados dchaS, izdaS, dchaG, izdaG cuando sean necesarios, o incluso dchaSG, izdaSG, dchaSizdaG, izdaSdchaG.

Igual que en *control.vhd* al llegar a la última fila, en la salida del bloque suavizado o en la salida de histéresis es necesario pasar a un estado donde sel='0' o guardarceros='0' respectivamente. Para el caso del suavizado sirve co_addrs como indicador de que en la salida de suavizado.vhd se está procesando la penúltima fila pues, a partir del momento donde el último píxel sale de la ROM, co_addrs se activa (recuérdese que se mantiene activo hasta que se haga un reset del contador), y en este momento, en la salida del bloque de suavizado se encuentra el píxel correspondiente a WIDTH+5 ciclos antes, es decir, se está procesando la penúltima fila. Es por ello por lo por lo que si el sistema se encuentra en algún estado correspondiente a dchaS (dchaS, dchaSG, dchaSizdaG) y co_addrs='1', se debe entrar en un estado en el que sel='0'.

Sin embargo, en el momento de la salida del último píxel de la ROM, en la salida de histeresis.vhd se encuentra el píxel correspondiente a 4*WIDTH+13 ciclos antes, es decir, se necesita esperar 4 filas para estar en la última fila. Para llevar la cuenta de estas 4 filas se utilizan 2 contadores. cont_1 y cont_adicional. El objetivo de los contadores es el siguiente: cont_adicional cuenta hasta 4. Para imágenes 640x480 sería suficiente con este contador, pues el píxel correspondiente a 4*WIDTH+13 ciclos anterior corresponde con un píxel

perteneciente a la fila WIDTH-5, solo hace falta esperar 4 filas para llegar a la última. Sin embargo, para imágenes con un WIDTH<13 este píxel pertenece a la fila WIDTH-6. El cont_1 tiene como objetivo que co_1 esté activo a partir del momento en el que se inicia el estado GyUl (esto se consigue introduciendo en el enable de cont_1 una señal: en_1a, proporcionada por el bloque de control que se activa en todos los estados dchaS (dchaS, dchaSG, dchaSizdaG) sumada en una puerta and con co_addrs. De esta forma, a partir de que co_addrs='1', cuando se entre en algún estado que contenga dchaS, co_1 se activará). Así, esta señal co_1 sumada con co_grad en una puerta and se introducirá como enable en cont_adicional y se contará las veces que se acaba una fila en el bloque histéresis. Cuando este contador llegue a 4, se estará procesando la penúltima fila en el bloque histeresis.vhd, por lo que se tendrá que pasar al estado Ultima, donde se guardan 0 en la memoria (guardarceros='0').

4.3.11 CONTADORES

- **CONT_ADDRS.VHD**

  Este bloque es idéntico al desarrollado para el algoritmo Sobel. Para comprender su funcionamiento véase *4.2.7 cont_addrs.vhd.*

- **CONT_CEROS.VHD**

  Este bloque es idéntico al desarrollado para el algoritmo Sobel. Para comprender su funcionamiento véase *4.2.8 cont_ceros.vhd.*

- **CONT_GRAD.VHD**

  Este bloque es idéntico al desarrollado para el algoritmo Sobel. Para comprender su funcionamiento véase *4.2.9cont_grad.vhd.*

- **CONT_SUAVIZADO.VHD**

  Este bloque tiene la misma funcionalidad que cont_grad, por lo que el desarrollo es idéntico. Para comprender su funcionamiento véase *4.2.9 cont_grad.vhd*.

- **CONT_1.VHD**

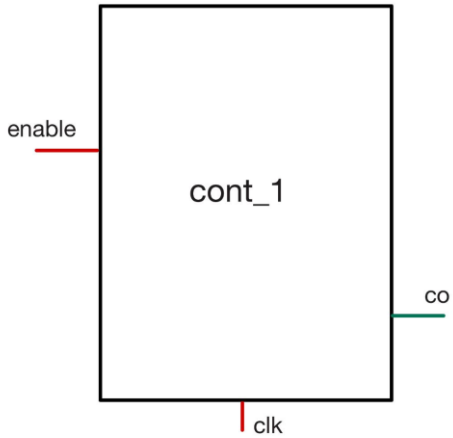


Figura 33. Bloque cont_1 en VHDL.

Se trata de un bloque que cuenta hasta 1 y cuando acaba la cuenta no se reinicia, y mantiene el valor del rebose a 1. Es decir, si su enable se encuentra activo en un ciclo de reloj, co se activa hasta que se haga un reset del contador. El código correspondiente se encuentra en el Anexo de códigos: *Código 28*.

- **CONT_ADICIONAL.VHD**

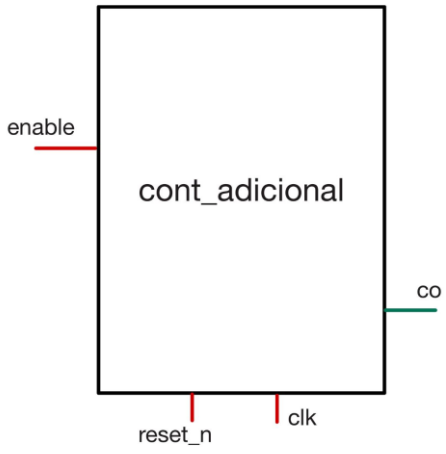


Figura 34. Bloque cont_adicional.vhd en VHDL

Es un contador que cuenta hasta 4 con habilitación de cuenta y rebose [18]. El código correspondiente es *Código 29.*

4.3.12 CANNY.VHD

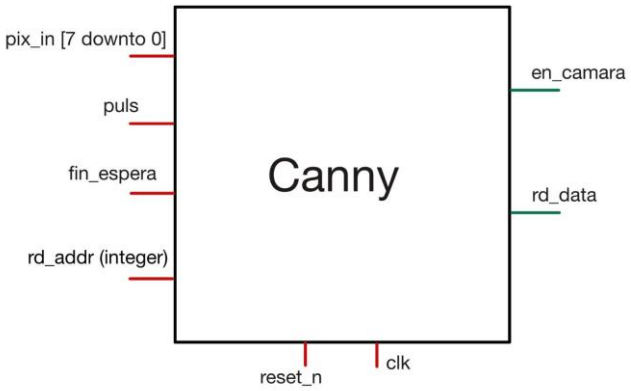


Figura 35. Bloque canny.vhd en VHDL

Este bloque es el bloque más alto de la jerarquía. Es un bloque estructural que establece las conexiones necesarias entre el resto de los bloques. El diagrama de conexión final es el que se observa en la *Figura 36*. Tiene como entradas y salidas las señales necesarias para la conexión del sistema con el exterior. El *Código 30* es el correspondiente a este bloque.

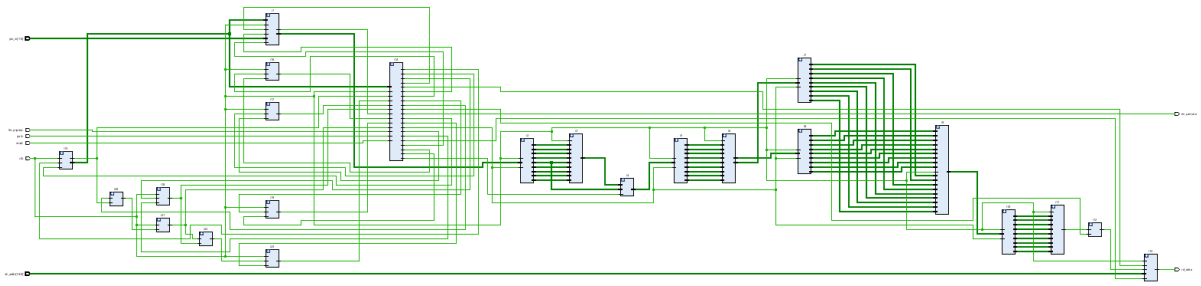


Figura 36. Estructura general de Canny en VHDL

Para mayor claridad de la imagen, se ha separado la imagen en 2 y se ha hecho zoom, que permite observar el nombre de las señales y bloques: *Figura 37* y *Figura 38*.

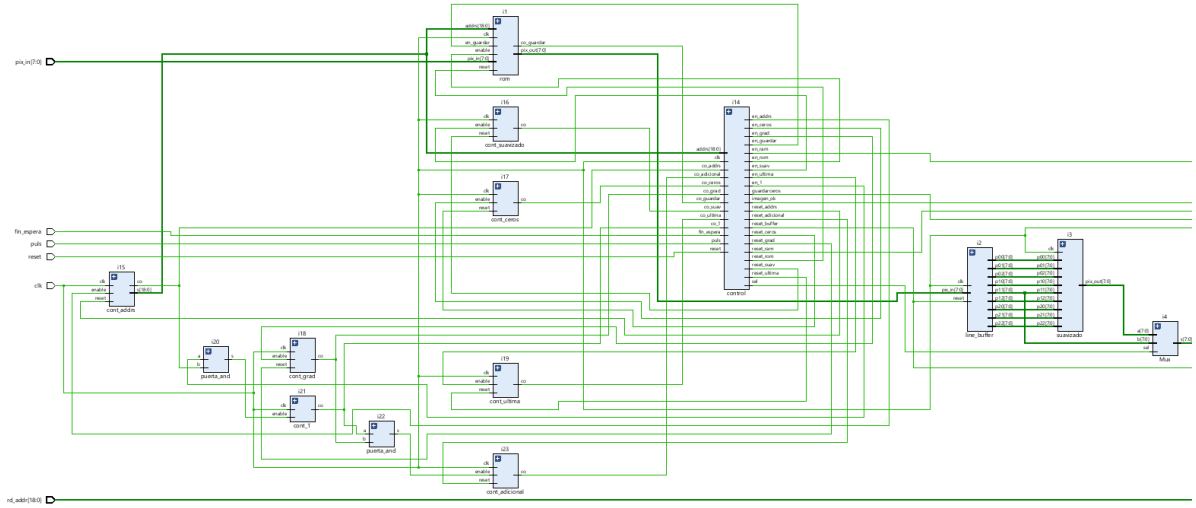


Figura 37. Zoom1 de la estructura general de Canny en VHDL

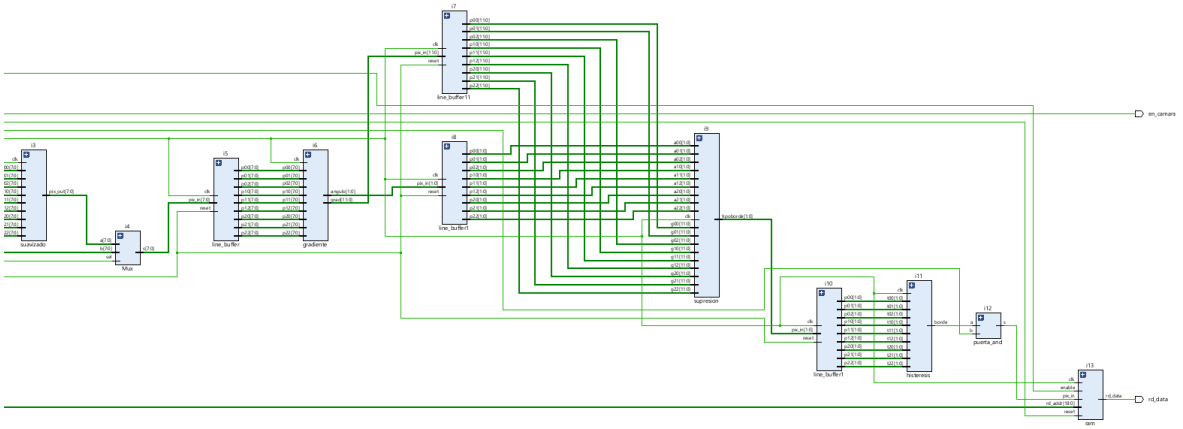


Figura 38. Zoom2 de la estructura general de Canny en VHDL

4.3.13 SIMULACIÓN

En esta sección se mostrará la simulación del algoritmo Canny con un enfoque mas general, pues el funcionamiento es muy parecido al mostrado en la simulación del proyecto Sobel. El objetivo es simplemente comprobar el correcto funcionamiento del algoritmo. El testbench diseñado es exactamente igual que el del algoritmo Sobel. El código de este testbench es el *Código 32.*

El resultado de la simulación para una imagen 10x10 con una línea de píxeles negros en la columna 3 es el siguiente:

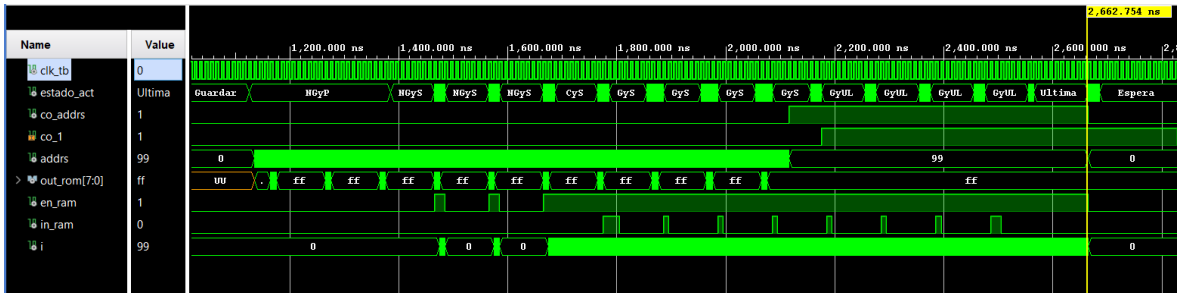
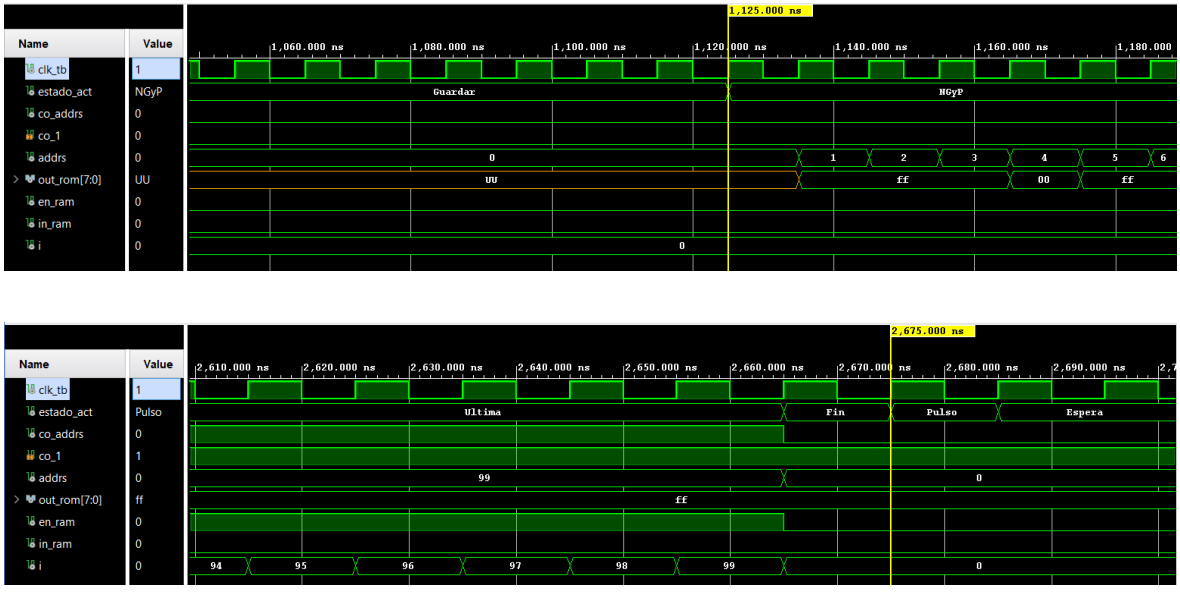


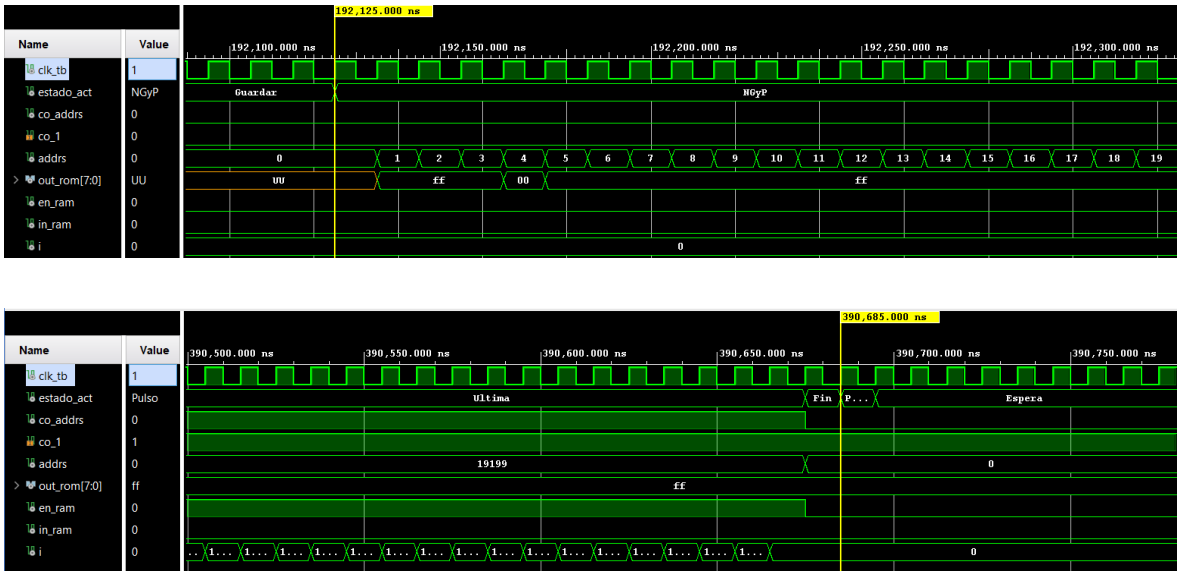
Figura 39. Simulación de una imagen 10x10 de Canny

Se comprueba que en torno al píxel 3 de cada fila se introduce un 1 en la RAM, es decir, un borde.

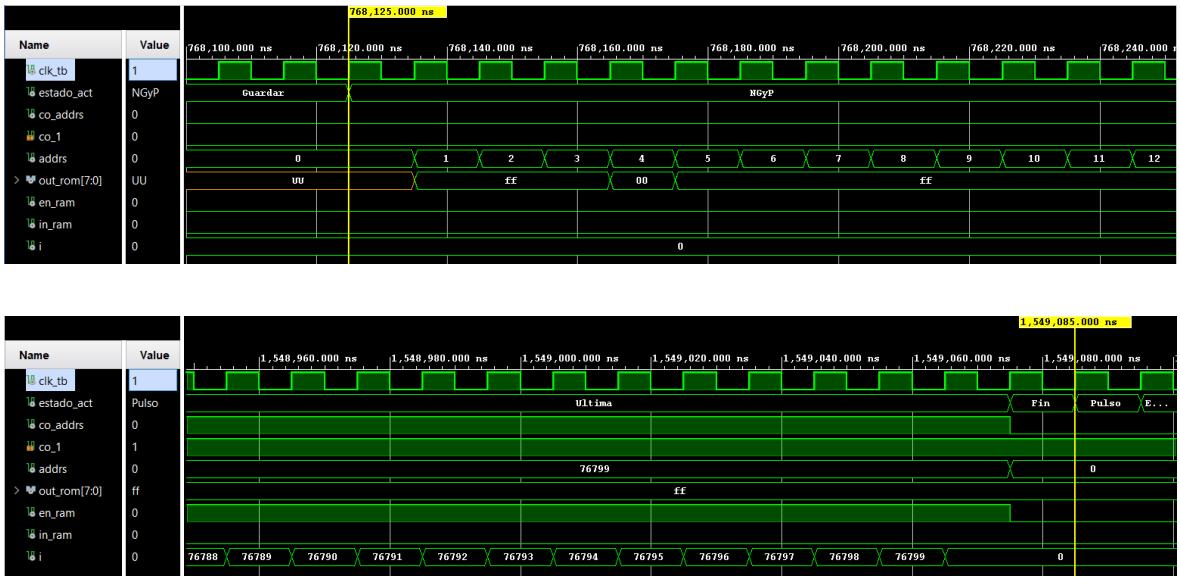
El tiempo de detección comienza en 1.125ns y acaba en 2.675ns:



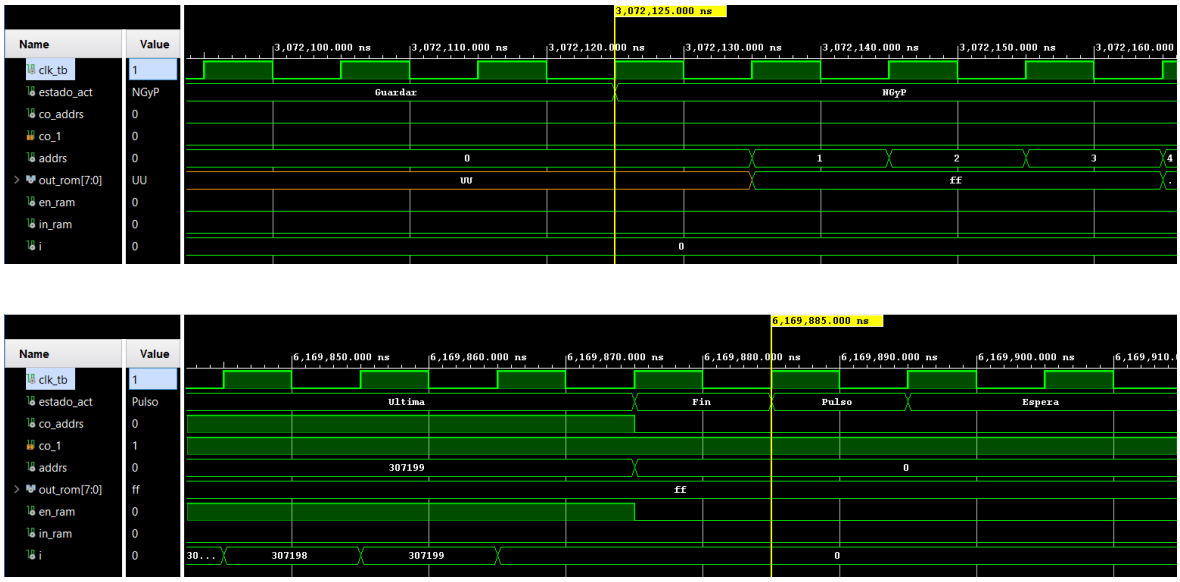
Para una imagen de 160x120, el tiempo de detección comienza en 192.195ns y finaliza en 390.685ns:



Para una imagen de 320x240, el tiempo de detección comienza en 768.125ns y finaliza en 1.549.085ns:



Para una imagen de 640x480 el tiempo de detección comienza en 3.072.125ns y finaliza en 6.169.885ns:



4.4 IMPLEMENTACIÓN FÍSICA Y VISIÓN FINAL DEL PROYECTO

Se ha implementado el proyecto en una FPGA habiendo cargado previamente distintas imágenes en la ROM. Las imágenes de prueba son imágenes de tamaño 320 x 240. Estas imágenes se convierten a un archivo que contiene todos los píxeles en bytes de 8 bits por píxel mediante un código de Python y se cargan en la ROM.

Para la implementación se ha diseñado el archivo de constraints necesario. Se ha realizado la síntesis, que convierte el código en un circuito, y posteriormente la implementación. Por último, se ha generado el archivo bitstream, que es el que la FPGA comprende para ponerse en funcionamiento.

Se obtienen los siguientes resultados a través de la representación del mapa de bordes:

4.4.1 SOBEL

Para el algoritmo Sobel se han probado las imágenes con diferentes valores en el umbral, para observar la precisión de detección dependiendo del umbral

Primera imagen:

- Imagen original

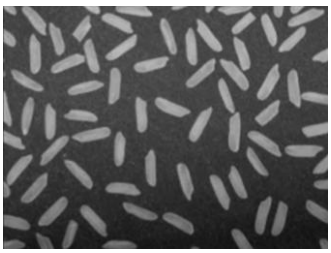


Figura 40. Imagen original de prueba 1

- Umbral=115

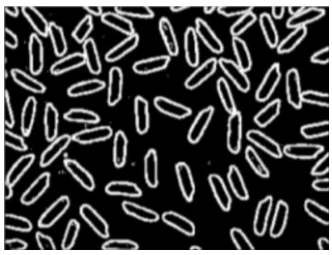


Figura 41. Mapa de bordes de Sobel sobre la imagen 1. Umbral=115

- Umbral=300

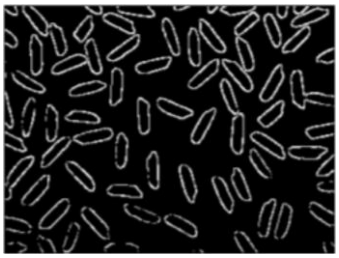


Figura 42. Mapa de bordes de Sobel sobre la imagen 1. Umbral=300

- Umbral=390

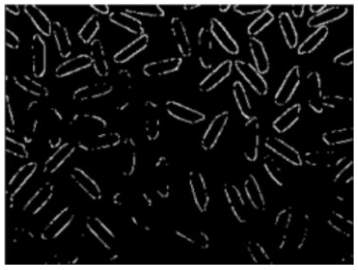


Figura 43. Mapa de bordes de Sobel sobre la imagen 1. Umbral=390

Segunda imagen:

- Imagen original

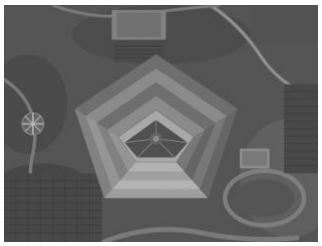


Figura 44. Imagen original de prueba 2

- Umbral=30

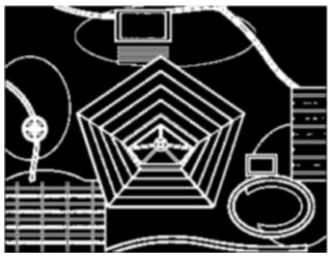


Figura 45. Mapa de bordes de Sobel sobre la imagen 2. Umbral=30

- Umbral=85

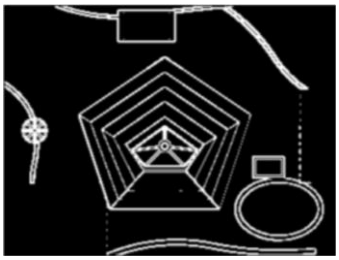


Figura 46. Mapa de bordes de Sobel sobre la imagen 2. Umbral=85

- Umbral=165

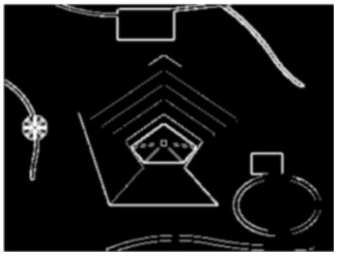


Figura 47. Mapa de bordes de Sobel sobre la imagen 2. Umbral=165

Tercera imagen:

- Imagen original

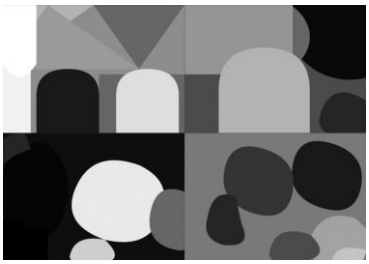


Figura 48. Imagen original de prueba 3

- Umbral=5

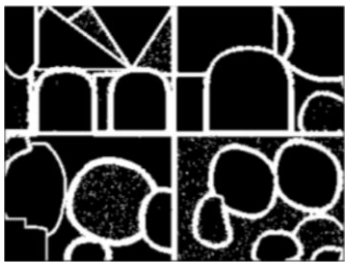


Figura 49. Mapa de bordes de Sobel sobre la imagen 3. Umbral=5

- Umbral=45

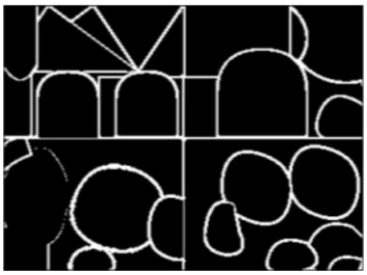


Figura 50. Mapa de bordes de Sobel sobre la imagen 3. Umbral=45

- Umbral=235

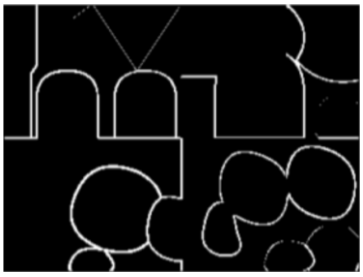


Figura 51. Mapa de bordes de Sobel sobre la imagen 3. Umbral=235

Cuarta imagen:

- Imagen original

Figura 52. Imagen original de prueba 4.

- Umbral=240

Figura 53. Mapa de bordes de Sobel sobre la imagen 4. Umbral=240

- Umbral=435

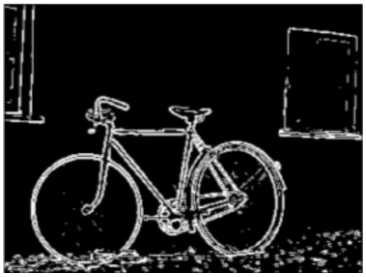


Figura 54. Mapa de bordes de Sobel sobre la imagen 4. Umbral=435

- Umbral=660

Figura 55. Mapa de bordes de Sobel sobre la imagen 4. Umbral=660

4.4.2 CANNY

Para el algoritmo Canny no se han cambiado los umbrales, se han hecho todas las pruebas para Talto=200 y Tbajo=80. De esta forma se obtiene:

Primera imagen:

- Imagen original:

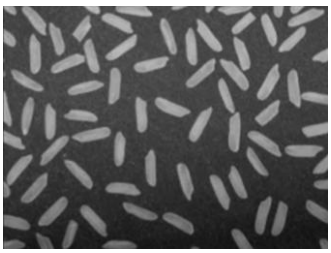


Figura 56. Imagen original de prueba 1 (canny)

- Mapa de bordes:

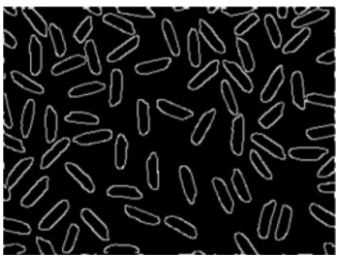


Figura 57. Mapa de bordes imagen 1 (canny)

Segunda imagen:

- Imagen original:

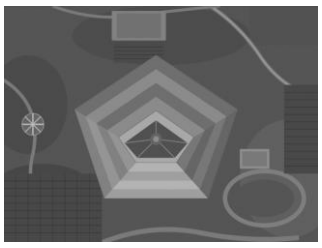


Figura 58. Imagen original de prueba 2 (canny)

- Mapa de bordes:

Figura 59. Mapa de bordes imagen 2 (canny)

Tercera imagen:

- Imagen original:

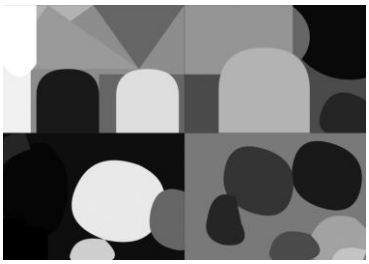


Figura 60. Imagen original de prueba 3. (canny)

- Mapa de bordes:

Figura 61. Mapa de bordes imagen 3 (canny)

Cuarta imagen:

- Imagen original:

Figura 62. Imagen original de prueba 4 (canny)

- Mapa de bordes:

Figura 63. Mapa de bordes imagen 4 (canny)

Capítulo 5. ANÁLISIS DE RESULTADOS

En este apartado se analizarán los resultados obtenidos de las simulaciones y pruebas físicas de los algoritmos Sobel y Canny, tanto en su implementación hardware como software. Se llevará a cabo una comparación de estas dos últimas implementaciones a través del análisis de las siguientes características: Rapidez en la detección, calidad en la detección, potencia consumida en la detección y análisis económico.

Se contemplan 2 escenarios de implementación de los algoritmos:

- FPGA Artix-7 XC7A35T (placa Basys-3) a 100 MHz, con el pipeline VHDL que procesa un píxel por ciclo de reloj y aritmética entera.
- Raspberry Pi 4B (CPU ARM Cortex-A72 de cuatro núcleos a 1,5 GHz, 4 GB de RAM)

Para evaluar y comparar las características de los resultados, ambas implementaciones son idénticas. Es decir, la implementación software en la Raspberry pi es una imitación y resuelve exactamente el mismo problema que el código desarrollado en VHDL:

- Mismas aproximaciones (G=|Gx|+|Gy|, ángulo) y mismo mecanismo de histéresis.
- Mismos umbrales y resoluciones: QVGA 320 × 240 y VGA 640 × 480

La Raspberry Pi será la pieza central de esta comparación: es un sistema embebido real, alimentado por USB, que se puede colocar al lado de la FPGA y medir en las mismas condiciones.

Las medidas para realizar la comparativa se han realizado para dos resoluciones:

- QVGA 320 x 240: A través de la implementación física en las placas.
- VGA 640 x 480: A través de una simulación.

5.1 RAPIDEZ EN LA DETECCIÓN

5.1.1 HARDWARE

Durante la simulación de los algoritmos Sobel y Canny en VHDL se ha explicado que, a través del conteo del número de ciclos de reloj que tarda la simulación en completar la detección se puede calcular cuánto tiempo tardará la FPGA en realizar una detección completa.

Si se observan los resultados obtenidos en la *sección 4.2.11* se puede obtener la siguiente tabla que refleja los tiempos de guardado y detección del algoritmo Sobel según la resolución de la imagen:

Resolución de la imagen	Tiempo de guardado (ns)	Tiempo de detección (ns)
10 x 10	1.125	1.160
160 x 120	192.125	193.660
320 x 240	768.125	771.260
640 x 480	3.072.125	3.078.460

Tabla 2. Tiempos de guardado y detección en imágenes de diferentes resoluciones en Sobel

Si se trazan las funciones de tiempo de guardado y tiempo de detección en función del número de píxeles total de la imagen se obtiene lo siguiente:

Figura 64. Gráfico del tiempo de detección y de guardado en función del tamaño de la imagen en Sobel

Se calcula el **tiempo de detección por píxel**:

- 10 x 10:

$$\frac{1.160}{100} = 11{,}6\ ns$$

- 160 x 120:

$$\frac{193.660}{19.200} = 10{,}086\ ns$$

- 320 x 240:

$$\frac{771.260}{76.800} = 10{,}042\ ns$$

- 640 x 480:

$$\frac{3.078.460}{307.200} = 10{,}021\ ns$$

Así mismo, si observan los resultados obtenidos en la *sección 4.3.13* se puede obtener la siguiente tabla que refleja los tiempos de guardado y detección del algoritmo Canny según la resolución de la imagen:

Resolución de la imagen	Tiempo de guardado (ns)	Tiempo de detección (ns)
10 x 10	1.125	1.550
160 x 120	192.125	198.560
320 x 240	768.125	780.960
640 x 480	3.072.125	3.097.760

Tabla 3. Tiempos de guardado y detección en imágenes de diferentes resoluciones en canny

Si se trazan las funciones de tiempo de guardado y tiempo de detección en función del tamaño de la imagen se obtiene lo siguiente:

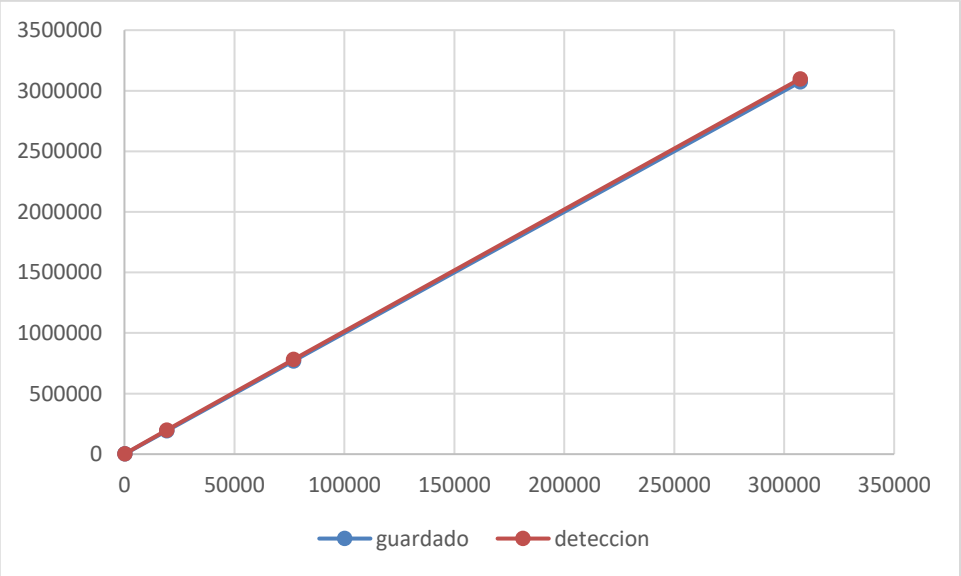


Figura 65. Gráfico del tiempo de detección y de guardado en función del tamaño de la imagen en Canny

Se calcula el **tiempo de detección por píxel**:

- 10 x 10:

$$\frac{1.550}{100} = 15{,}5\ ns$$

- 160 x 120:

$$\frac{198.560}{19.200} = 10{,}34\ ns$$

- 320 x 240:

$$\frac{780.960}{76.800} = 10{,}17\ ns$$

- 640 x 480:

$$\frac{3.097.760}{307.200} = 10{,}08\ ns$$

Se observa en las gráficas: *Figura 64* y *Figura 65*, un crecimiento lineal en el tiempo de detección y de guardado con el tamaño de la imagen. Esto es una ventaja, pues, para imágenes de gran tamaño, el tiempo de detección aumenta con el número de píxeles.

El tiempo de detección por píxel en la implementación FPGA es de aproximadamente un ciclo de reloj (10ns). Es decir, se tiene un throughput de 1 píxel por ciclo aproximadamente (100Mpix/s). Se excede un poco debido al retardo en el inicio del pipeline que conllevan estos sistemas: WIDTH+4 en el caso de Sobel y 4*WIDTH+13 ciclos en el caso de Canny. Por lo tanto, si se considera un throughput de 1 píxel por ciclo (el tiempo de detección por píxel tendría que ser 10ns), el **tiempo de arranque por frame** obtenido a partir de la simulación es de (se compara también con el resultado teórico de latencia):

<u>*SOBEL:*</u>

- 10 x 10:

$$(11{,}6ns - 10ns) * 100 = 160\ ns \sim 10 * (10 + 4)$$

- 160 x 120:

$$(10{,}086ns - 10ns) * 19.200 = 1651\ ns \sim 10 * (160 + 4)$$

- 320 x 240:

$$(10{,}042ns - 10ns) * 76.800 = 3225\ ns \sim 10 * (320 + 4)$$

- 640 x 480:

$$(10{,}021ns - 10ns) * 307.200 = 6.451\ ns \sim 10 * (640 + 4)$$

CANNY:

- 10 x 10:

$$(15{,}5ns - 10ns) * 100 = 550\ ns \sim 10 * (4 * 10 + 13)$$

- 160 x 120:

$$(10{,}34ns - 10ns) * 19.200 = 6.528\ ns \sim 10 * (4 * 160 + 13)$$

- 320 x 240:

$$(10{,}17ns - 10ns) * 76.800 = 13.056\ ns \sim 10 * (4 * 320 + 13)$$

- 640 x 480:

$$(10{,}08ns - 10ns) * 307.200 = 24.576\ ns \sim 10 * (4 * 640 + 13)$$

Otra característica interesante son los **FPS** que el sistema alcanza. La FPGA tiene integrado un reloj de 100MHz, por lo tanto:

- 10 x 10:

$$\frac{100 * 10^6}{100} = 1.000.000\ FPS$$

- 160 x 120:

$$\frac{100 * 10^6}{19.200} = 15.208{,}33\ FPS$$

- 320 x 240:

$$\frac{100 * 10^6}{76.800} = 1.302\ FPS$$

- 640 x 480:

$$\frac{100 * 10^6}{307.200} = 325{,}5\ FPS$$

5.1.2 SOFTWARE

Al implementar el sistema de detección de bordes Canny en la Raspberry pi, realizando las medidas pertinentes se obtienen los siguientes resultados en relación con el tiempo de detección:

QVGA 320 x 240

Plataforma	Latencia/frame(ms)	FPS	Mpix/s
Raspberry pi (1 núcleo)	36,17	27,65	2,12
Raspberry pi (4 núcleos)	31,44	31,81	2,44

Tabla 4. Resumen de la eficacia en la detección software QVGA 320 x 240

VGA 640 x 480

Plataforma	Latencia/frame(ms)	FPS	Mpix/s
Raspberry pi (1 núcleo)	125,93	7,94	2,44
Raspberry pi (4 núcleos)	90,5	11,05	3,39

Tabla 5. Resumen de la eficacia en la detección software VGA 640 x 480

El paso a 4 núcleos tiene una ganancia menor que 4:

	Speed-up medido
QVGA 320 x 240	$\frac{2,44}{2,12} = 1,15$
VGA 640 x 480	$\frac{3,39}{2,44} = 1,39$

Tabla 6. Speed-up con 4 núcleos

Esto puede deberse a que el coste de disponer y repartir la carga de la imagen entre los 4 núcleos es muy pesado cuando el cálculo por banda ya es rápido, sobre todo en imágenes pequeñas (de ahí que QVGA escale peor que VGA).

Por otro lado, se ha medido el tiempo de cada etapa del algoritmo por separado y se obtiene lo siguiente:

Etapa	**Tiempo (ms)**	**% del total**
Suavizado	8,24	23,1%
Gradiente	18,07	50,7%
Supresión de no máximos	7,98	22,4%
Histéresis	1,33	3,7%

Tabla 7. Tiempo que pasa el algoritmo en cada etapa

El algoritmo se pasa la mayor parte del tiempo en la etapa del cálculo del gradiente (50,7%). Para imágenes de mayor resolución este porcentaje todavía se incrementa más. Esta etapa es precisamente la que, para una FPGA tiene un coste constante (un píxel por ciclo), que no aumenta con la resolución.

5.1.3 COMPARATIVA

Por último y como comparativa final en la rapidez de detección, se comparan cada una de las características (Se contempla únicamente el algoritmo Canny):

QVGA 320 x 240

Plataforma	Latencia/frame(ms)	FPS	Mpix/s
FPGA	0,78	1.302	100
Raspberry pi (1 núcleo)	36,17	27,65	2,12
Raspberry pi (4 núcleos)	31,44	31,81	2,44

Tabla 8. Resumen y comparativa características de rapidez en la detección FPGA y raspberry (QVGA 320 x 240)

VGA 640 x 480

Plataforma	Latencia/frame(ms)	FPS	Mpix/s
FPGA	3,09	326	100
Raspberry pi (1 núcleo)	125,93	7,94	2,44
Raspberry pi (4 núcleos)	90,5	11,05	3,39

Tabla 9. Resumen y comparativa características de rapidez en la detección FPGA y raspberry (VGA 640 x 480)

A partir de esta información se pueden sacar las siguientes conclusiones:

1. La FPGA tiene un throughput constante (100 Mpix/s), independiente del tamaño de la imagen. Mientras tanto, la Raspberry Pi tiene un throughput de 2,1-3,4 Mpix/s, mucho inferior a la FPGA. Además, el tiempo aumenta con la resolución. Fases como el gradiente, que ocupan la mayor parte del espacio en el desarrollo en la Raspberry Pi, e incrementan su coste con el aumento del tamaño de la imagen, en la FPGA mantienen un coste constante: un píxel por ciclo. Esto justifica directamente la decisión de acelerar en hardware para imágenes de gran tamaño
2. Sin embargo, la Raspberry Pi mantiene un tiempo real holgado (11-30 FPS).
3. Añadir núcleos a una CPU no es equivalente a la paralelización masiva y sin sobrecoste que ofrece el pipeline hardware. Debido al costoso proceso de separación y distribución de la información, y otros factores, no siempre es rentable (o no tanto como se esperaría) añadir núcleos. Mientras tanto, el pipeline de la FPGA cuenta con un paralelismo intrínseco y sin sobrecoste.

5.2 CALIDAD DE LA DETECCIÓN

Tanto el código Python como la simulación VHDL leen exactamente los mismos píxeles (el mismo fichero de bytes). Así, cualquier diferencia en el resultado solo puede deberse al procesado, nunca a que partan de imágenes distintas. Para realizar una comparativa entre los resultados de las dos implementaciones se introducen los resultados en un código Python que devuelve un mapa de bordes diferencial que indica que pixeles son diferentes en ambos resultados:

- **SOBEL**

Primera imagen:

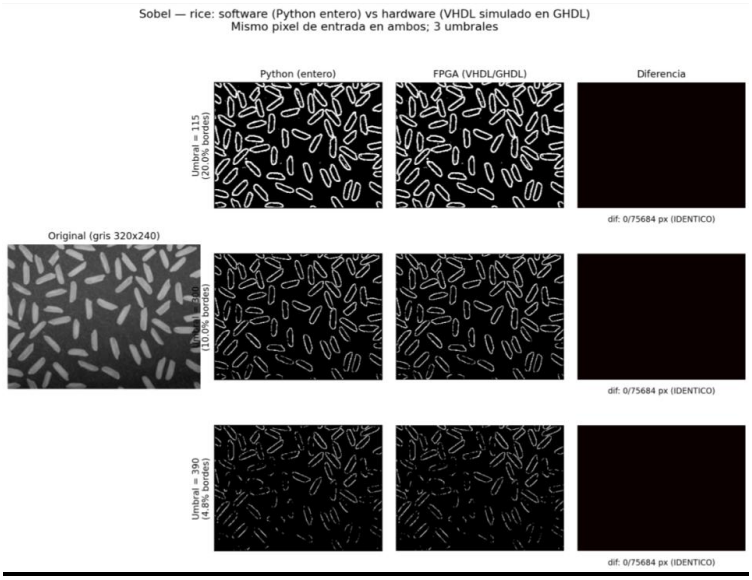


Figura 66. Comparación mapa de bordes en sobel entre implementación software y hardware de la imagen 1

Segunda imagen:

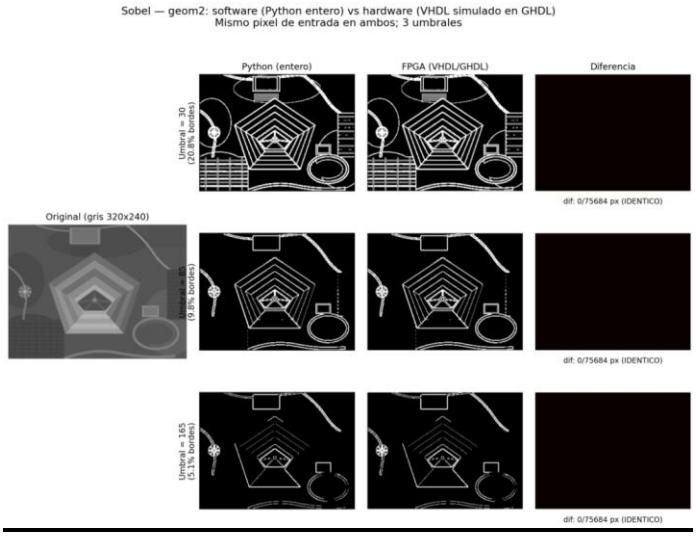


Figura 67. Comparación mapa de bordes en sobel entre implementación software y hardware de la imagen 2

Tercera imagen:

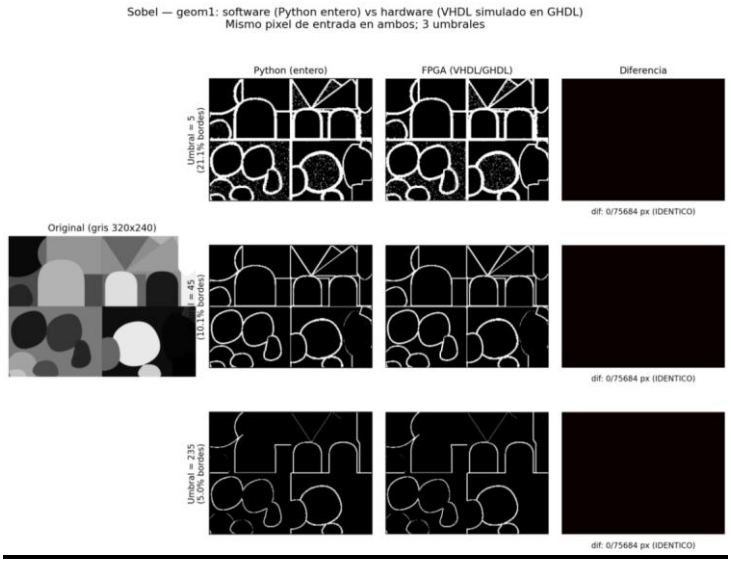


Figura 68. Comparación mapa de bordes en sobel entre implementación software y hardware de la imagen 2

Cuarta imagen:

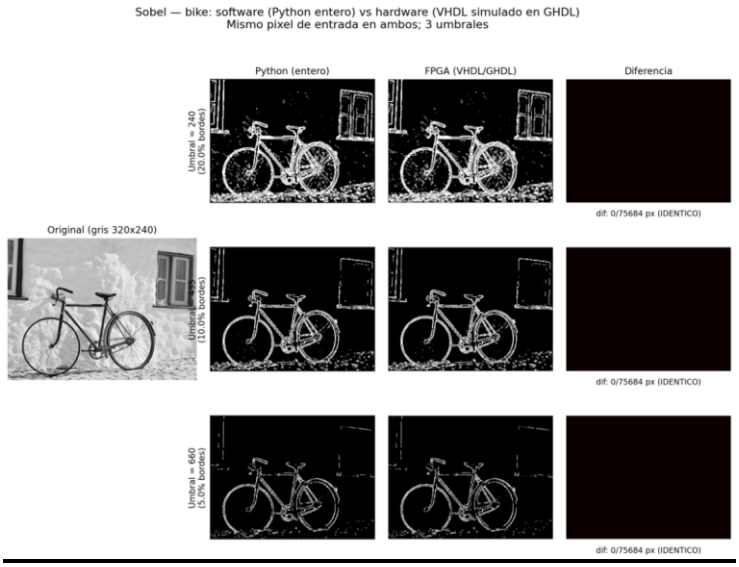


Figura 69. Comparación mapa de bordes en sobel entre implementación software y hardware de la imagen 4

- **CANNY**

Primera imagen:

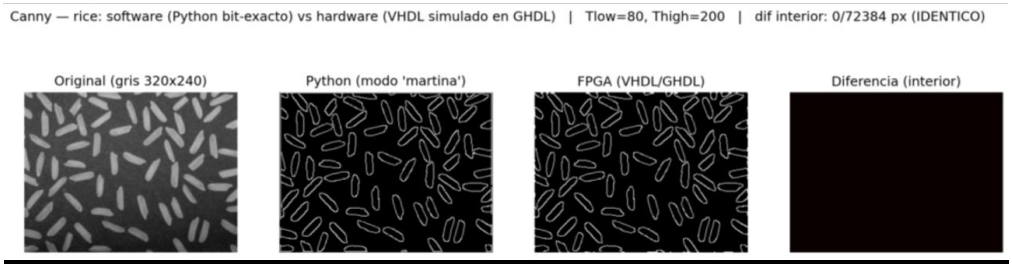


Figura 70. Comparación mapa de bordes en canny entre implementación software y hardware de la imagen 1

Segunda imagen:

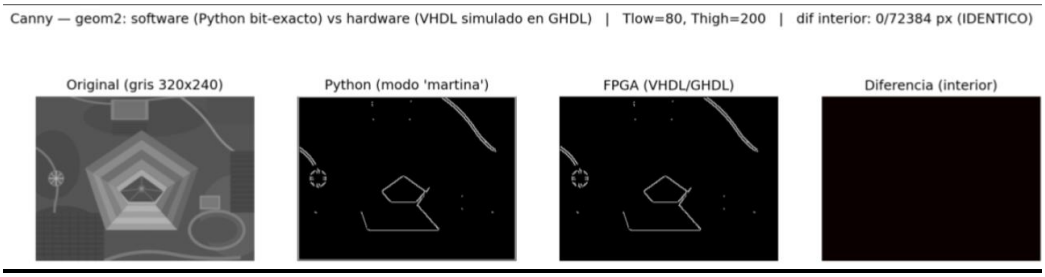


Figura 71. Comparación mapa de bordes en canny entre implementación software y hardware de la imagen 2

Tercera imagen:

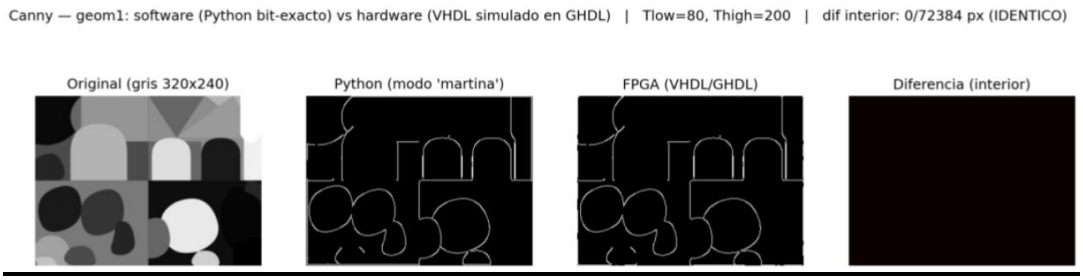


Figura 72. Comparación mapa de bordes en canny entre implementación software y hardware de la imagen 3

Cuarta imagen:

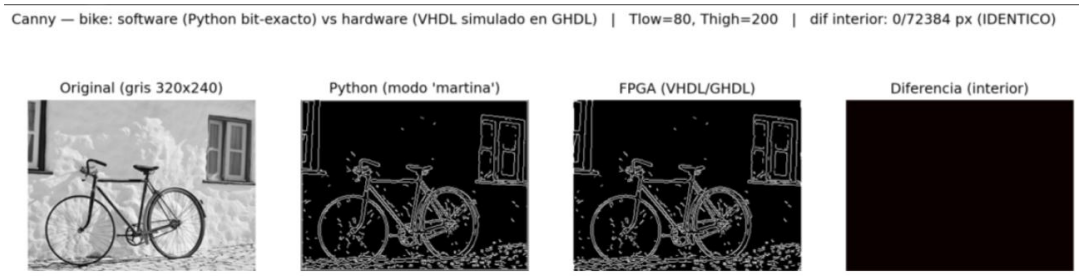


Figura 73. Comparación mapa de bordes en canny entre implementación software y hardware de la imagen 4

En absolutamente todas las comparaciones el mapa de comparación generado es una imagen en negro. Esto quiere decir que ambas implementaciones ofrecen un mapa de bordes idéntico. Todos los píxeles son idénticos. Por lo tanto, en cuanto a la calidad de la detección, ambas implementaciones tienen la misma eficiencia.

5.3 ANÁLISIS ENERGÉTICO

5.3.1 HARDWARE

Para el cálculo de la potencia consumida por los algoritmos cuando se implementan en una FPGA se ha alimentado la FPGA con un circuito externo. La Basys-3 ofrece esta posibilidad, en lugar de la alimentación a través del cable USB que se conecta al ordenador. Simplemente hay que cambiar el jumper a la posición que se observa en la *Figura 74* y conectar el circuito de alimentación a los pines EXT y GND.

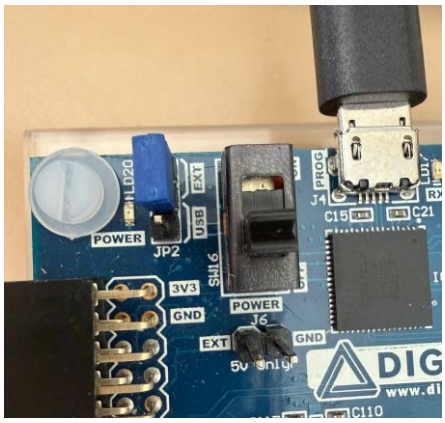


Figura 74. Posición del jumper para alimentación externa

El circuito de alimentación conectado es el siguiente:

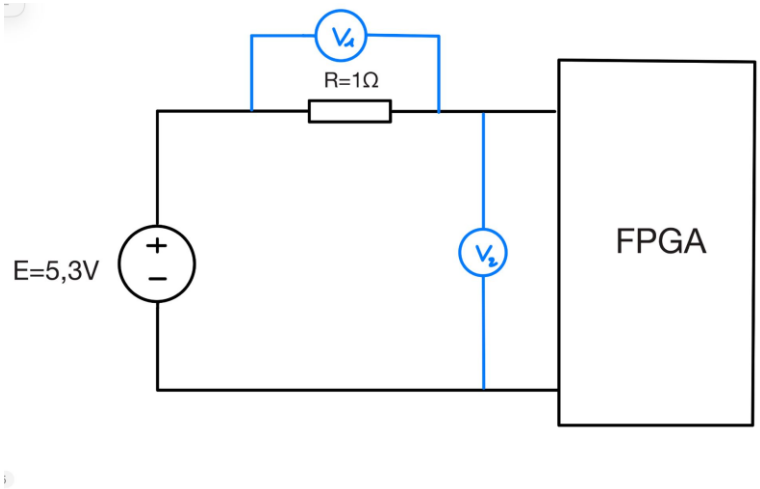


Figura 75. Circuito de alimentación FPGA.

De esta forma, los resultados obtenidos al implementar el programa en la FPGA (el programa detecta bordes en bucle) y alimentarla con este circuito son:

La medida V1 es la siguiente:

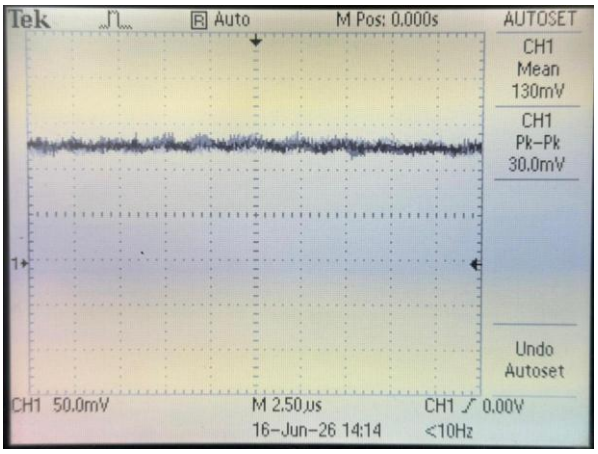


Figura 76. Caída de tensión en la resistencia del circuito de alimentación de la FPGA mientras se procesa Sobel

Con esto podemos obtener la corriente que circula por la resistencia, y por lo tanto la que se introduce en la FPGA:

$$\frac{130\ mV}{1\ \Omega} = 130\ mA$$

La medida V2 es:

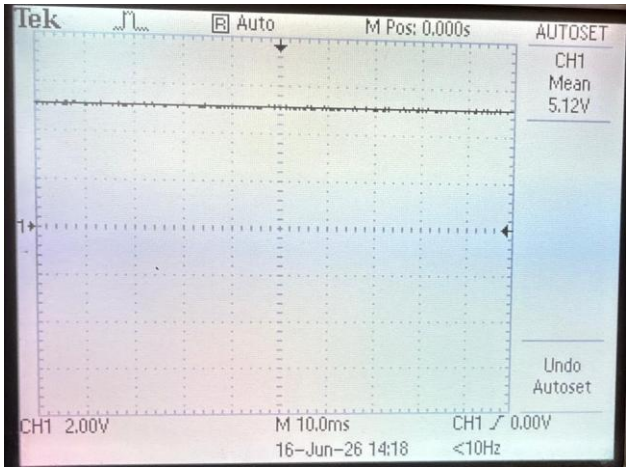


Figura 77. Tensión de alimentación a la FPGA mientras se procesa Sobel

La potencia resultante que consume la FPGA es de:

$$P = 5{,}12\ V * 0{,}13\ A = \mathbf{0,666\ W}$$

A partir de este valor y conociendo el throughput constante que maneja la FPGA: 100 Mpix/s, y los FPS: 326, se pueden calcular las siguientes características:

$$\frac{J}{Mpix} = \frac{0{,}666\ W}{100\ MPix/s} = 0{,}00666\ J/Mpix$$

$$\frac{J}{frame} = \frac{0{,}666\ W}{326\ frame/s} = 0{,}002\ J/frame$$

5.3.2 SOFTWARE

La Raspberry Pi no incorpora un sensor de potencia que permita leer directamente el consumo. Es por ello por lo que se ha realizado una estimación:

	P reposo (W)	**P carga (W)**	**J/frame**	**J/Mpix**
Raspberry Pi	3	6,4	0,58-0,82	1,89-2,67

Tabla 10. Consumo energético Raspberry Pi

Los valores de 3,0 W (reposo) y 6,4 W (carga) se han tomado de documentación. Sí que se han comprobado las siguientes medidas:

- La CPU estaba trabajando al 94,5 %
- La temperatura era de 65 °C
- No había throttling (throttled = 0x0), es decir, la Raspberry no redujo su frecuencia por temperatura.

Es por ello por lo que se considera razonable utilizar el valor típico de potencia en carga (~6,4 W) para estimar la energía consumida.

5.3.3 COMPARATIVA

Con los resultados obtenidos después de las dos implementaciones se construye la siguiente tabla:

	P reposo (W)	P carga (W)	J/frame	J/Mpix
FPGA	0,5	0,666	0,002	0,0066
Raspberry Pi	3	6,4	0,58-0,82	1,89-2,67

Tabla 11. Comparación del consumo energético

De esta manera se confirma que la FPGA es una solución mucho más eficiente energéticamente, alcanzando un consumo 10 veces menor que la Raspberry Pi (0,66 W frente a 6,4 W). En términos de energía por píxel, la FPGA es 300 veces mejor que la Raspberry Pi (0,0066 J/Mpix frente a 2 J/Mpix).

5.4 ANÁLISIS ECONÓMICO

Por último, para cerrar la comparativa es necesario llevar a cabo un análisis económico de cada uno de los sistemas. Durante este análisis se detallarán las siguientes características: Coste de inversión, coste de desarrollo de software, coste de operación (energético), coste normalizado y vida útil.

5.4.1 COSTE DE INVERSIÓN

El coste de inversión refleja el desembolso inicial requerido para que sea posible la puesta en funcionamiento del sistema. En este trabajo, dicho coste incluye la adquisición del hardware requerido para implementar y validar los algoritmos de detección de bordes en cada una de las plataformas estudiadas, tanto la solución basada en FPGA como la basada en Raspberry Pi.

En la ruta FPGA, los elementos necesarios y sus costes asociados se reflejan en la *Tabla 12*:

	Precio (€)
Placa Basys-3	150€ [24]
Circuito de alimentación y protoboard necesarias para el cálculo de la energía consumida	40€
Tarjeta SD y cables USB necesarios	15€
Monitor o pantalla VGA	200€

Tabla 12. Costes de inversión de la ruta FPGA

Se ha considerado una tarjeta microSD de 32 GB necesaria para el almacenamiento del sistema operativo, imágenes de prueba y resultados. A partir de los precios observados en distintos distribuidores, se adopta un coste medio de 10 €. Para la VGA necesaria para la representación de los resultados se ha estimado un coste de mercado de 200 €. No obstante, estos dos valores, al ser compartidos en la ruta FPGA y la ruta Raspberry Pi, no intervienen en la comparación.

La ruta Raspberry Pi contará con los siguientes costes de inversión ():

	Precio (€)
Raspberry pi	105 € [25]
Adaptador HDMI-VGA	10 €
Tarjeta SD y cables USB necesarios	15 €
Monitor o pantalla VGA	200 €

Tabla 13. Costes de inversión de la ruta Raspberry Pi

La Raspberry Pi no cuenta con un puerto VGA físicamente, como es el caso de la FPGA. Es por ello por lo que se necesita un adaptador para esta conexión. Estos adaptadores rondan un precio de 10 € en el mercado.

Así, los costes totales:

	Coste de inversión total (€)
FPGA	405
Raspberry Pi	330

Tabla 14. Comparativa costes de inversión.

El coste de inversión inicial de la FPGA es superior al coste de la Raspberry Pi.

5.4.2 COSTE DE DESARROLLO DE SOFTWARE

El coste de desarrollo representa el valor económico asociado al tiempo y los recursos humanos necesarios para diseñar, implementar, verificar y validar una solución tecnológica. En este trabajo, este coste, se estima a partir de las horas dedicadas en cada una de las implementaciones (FPGA y Raspberry Pi), empleando una tarifa horaria representativa de un ingeniero junior.

	Tiempo total dedicado al TFG	**Porcentaje del tiempo ocupado**	**Tiempo total dedicado**	**Salario ingeniero junior Esp.**	**Tarifa horaria ingeniero**	**Coste total**
Ruta FPGA	300h	70 %	210 h	30.000 €/año	20 €/h	4.200 €
Ruta Raspberry		30 %	90 h			1.800 €

Tabla 15. Comparativa costes de desarrollo.

Para estimar el coste de desarrollo se ha considerado una tarifa de 20 €/h correspondiente a un ingeniero de desarrollo de software junior. Este valor se ha obtenido a partir de salarios

de entrada del sector software en España publicados por portales especializados de empleo como Glassdoor [26] y Talent.com [27], que utilizan datos y estudian estadísticas reales para llegar a estos resultados. Estos portales aproximan el salario medio en España de un ingeniero de desarrollo de software junior en 24.000-30.000€ al año. Considerando una jornada laboral anual aproximada de 1.700 hora, este salario se transforma en una tarifa horaria de aproximadamente 20 €/h.

El coste de desarrollo de la FPGA es superior al de la ruta Raspberry Pi debido a la mayor complejidad de código.

5.4.3 COSTE DE OPERACIÓN

El coste de operación engloba los gastos asociados al funcionamiento de un sistema una vez que este ha sido desarrollado e implantado. Este coste se genera de forma constante a lo largo de la vida útil del sistema. En sistemas electrónicos este coste está directamente relacionado con el consumo energético. El coste de operación se calculará a partir de los datos obtenidos anteriormente (*Tabla 11*) y a través de una estimación del coste económico asociado al consumo eléctrico. Para este valor se ha estimado un coste de 0,26 €/kWh, pues son, aproximadamente, los resultados obtenidos a partir de estadísticas de Eurostat para el año 2025 [28]. De esta forma:

	Potencia en carga (W)	**Potencia por Mpix (J/Mpix)**	**Media coste económico (€/kWh)**	**Media coste económico (€/J)**	**Coste por Mpix (€/Mpix)**
FPGA	0,66	0,0066	0,26	$7{,}25\text{x}10^{-8}$	$4{,}8\text{x}10^{-10}$
Raspberry Pi	6,4	2			$1{,}45\text{x}10^{-7}$

Tabla 16. Comparativa coste de operación

En este caso el coste es menor para la FPGA. Esto es lógico al saber que la FPGA consume mucha menos energía que la Raspberry Pi.

5.4.4 VIDA ÚTIL

La vida útil de los sistemas se estima a través de las tasas de amortización y períodos máximos de la agencia tributaria para equipos electrónicos [29]. Se estima una vida útil de entre 5 y 10 años. También se puede estimar este parámetro suponiendo que la vida útil de los equipos electrónicos está limitada por la obsolescencia de la tecnología según [30]. En este aspecto la Raspberry Pi está mas expuesta a la obsolescencia, mientras que la FPGA no depende de un sistema operativo ni de actualizaciones software. De este modo se puede estimar que la vida útil de los sistemas:

	Vida útil (años)
FPGA	9
Raspberry Pi	6

Tabla 17. Comparativa vida útil de los dispositivos

5.4.5 COSTE NORMALIZADO

El coste normalizado permite relacionar el coste económico de un sistema con la cantidad de trabajo útil que este es capaz de realizar. Este apartado tiene el objetivo de la comparativa general y una visión más completa de la viabilidad económica de cada solución.

	Inversión inicial (€)	**Coste de operación (€/Mpix)**
FPGA	4.605	$4{,}8\text{x}10^{-10}$
Raspberry Pi	2.130	$1{,}45\text{x}10^{-7}$

Tabla 18. Comparativa costes normalizados

El análisis del coste de desarrollo muestra que la implementación hardware requiere una mayor dedicación temporal, lo que se traduce en un mayor coste de implementación. Este

resultado era esperable debido a la mayor complejidad inherente al desarrollo en VHDL, así como a las tareas adicionales de síntesis, implementación y verificación hardware.

Sin embargo, cuando se analiza el coste de operación asociado al consumo energético, la situación se invierte e obtiene un coste energético de $4{,}8 \cdot 10^{-10}$ €/Mpix para la FPGA y de $1{,}45 \cdot 10^{-7}$ €/Mpix para la Raspberry Pi. Esto implica que el coste de operación por megapíxel procesado es aproximadamente 303 veces menor en la FPGA.

Desde una perspectiva global, ambos presentan comportamientos diferentes. La FPGA supone una inversión inicial mayor, pero es más rentable en términos de costes de operación, lo que la convierte en un sistema apropiado para entornos de trabajo intensivos y continuados en el tiempo.

Sin embargo, para escenarios de uso ocasional o prototipado rápido, la Raspberry Pi constituye una alternativa económicamente favorable debido a su menor esfuerzo de desarrollo.

Capítulo 6. CONCLUSIONES Y TRABAJOS FUTUROS

En el presente proyecto se han diseñado los algoritmos de detección de bordes Sobel y Canny en VHDL para la implementación sobre una FPGA Artix-7 XC7A35T (placa Basys-3). Adicionalmente se ha implementado una réplica idéntica del diseño en Python, con el objetivo de una comparación justa, sobre un PC de desarrollo y sobre una Raspberry Pi 4B (CPU ARM Cortex-A72 de cuatro núcleos a 1,5 GHz, 4 GB de RAM).

Los resultados obtenidos son los que se observan en el *Capítulo 5: Análisis de Resultados.*

Se han obtenido las siguientes conclusiones:

1. Aumentar el tamaño de las imágenes en la Raspberry pi supone un mayor tiempo de procesado de las etapas, mientras que en la FPGA hay un throughput constante de 100Mpix/s.
2. La paralelización en la Raspberry Pi supone un sobrecoste, mientras que en la FPGA este paralelismo es intrínseco.
3. La FPGA es una solución mucho más eficiente energéticamente, alcanzando un consumo 10 veces menor que la Raspberry Pi (0,66 W frente a 6,4 W). En términos de energía por píxel, la FPGA es 300 veces mejor que la Raspberry Pi (0,0066 J/Mpix frente a 2 J/Mpix).
4. En términos económicos, la FPGA supone un mayor coste inicial, debido a la mayor complejidad en la lógica y por lo tanto, necesidad de una mayor dedicación temporal para la implantación. Sin embargo, en relación con el coste de operación la FPGA supone una solución mucho mas rentable que la Raspberry Pi. Esto propone una conclusión clara: En según qué condiciones sea necesario trabajar será mas rentable un sistema u otro.

A partir de estos resultados se deduce que las FPGA conforman una alternativa para la implementación de los algoritmos de detección de bordes con una eficiencia energética excelente y con unas características y rapidez de detección competentes. Bien es cierto que existen alternativas software con mejores prestaciones que la Raspberry Pi, energéticamente las FPGA siguen siendo la mejor solución.

BIBLIOGRAFÍA

[1] B. Lasheras Hernández, «Estudio de Sistemas Avanzados de Asistencia al Conductor (ADAS) en vehículos y propuesta de aplicación de técnicas de seguimiento de la mirada para su mejora,» Trabajo de Fin de Grado, Escuela de Ingeniería y Arquitectura (EINA), Universidad de Zaragoza, Zaragoza, España, 2021.

[2] V. Maksimovic, B. Jaksic, M. Milosevic, J. Todorovic y L. Mosurovic, «Comparative Analysis of Edge Detection Operators Using a Threshold Estimation Approach on Medical Noisy Images with Different Complexities,» *Sensors 2025,* vol. 25, nº 1, 2024.

[3] S. Ravichandran, H.-K. Su, W.-K. Kuo, D. Dhanasekaran, M. Mahalingam y J.-P. Yang, «Parallel Processing of Sobel Edge Detection on FPGA: Enhancing Real-Time Image Analysis,» *Sensors,* vol. 25, 2025.

[4] Y. S. Domingo, «Universidad Politécnica de Valencia,» [En línea]. Available: https://yosedo.webs.upv.es/curso_edicion_de_imagenes_nivel_basico/1_7.html. [Último acceso: 7 Junio 2026].

[5] Departamento de Ingeniería Electrónica, Telecomunicación y Automática de la Universidad de Jaén, «Detección de bordes en una imagen,» material docente de prácticas, Área de Ingeniería de Sistemas y Automática, Universidad de Jaén, Jaén, España, 2006. Available:https://www4.ujaen.es/~satorres/practicas/practica3_vc.pdf. [Último acceso: Junio 2026]

[6] A. Nosrat y Y. S. Kavian, «Hardware Description of Multi-Directional Fast Sobel Edge Detection Processor by VHDL for Implementing on FPGA,» *International Journal of Computer Applications,* pp. Hardware Description of Multi-Directional

Fast Sobel Edge Detection Processor by VHDL for Implementing on FPGA, Junio 2012.

[7] R. Millon, E. Frati y E. Rucci, «Implementacion de Filtro de Detección de Bordes en SoC usando Síntesis de Alto Nivel,» Agosto 2020. [En línea]. Available: https://sedici.unlp.edu.ar/bitstream/handle/10915/102579/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y. [Último acceso: Marzo 2026].

[8] J. Villemejane, «Lense Instititut Optique,» 2024. [En línea]. Available: https://iogs-lense-training.github.io/image-processing/contents/opencv_blur.html. [Último acceso: Junio 2026].

[9] J. V. Rebaza, «Detección de bordes mediante el algoritmo de Canny,» [En línea]. Available: https://www.researchgate.net/profile/Jorge-Valverde-Rebaza/publication/267240432_Deteccion_de_bordes_mediante_el_algoritmo_de_Canny/links/548dd1ae0cf225bf66a5f636/Deteccion-de-bordes-mediante-el-algoritmo-de-Canny.pdf. [Último acceso: Marzo 2026].

[10] J. Campos, J. Rimari-Leon, F. Huanasca-Romero, Q. Moratillo, S. Ore Orihuela, Z. Bermudez Mayta y L. X. Huaman Romero, «Discernimiento de bordes con el algoritmo Canny para la identificación y conteo de cuyes,» *Voz Zootecnista,* vol. 3, pp. 19-22, 2022.

[11] M. A. Jaramillo, Á. Fernández y E. Martínez de Salazar, «Implementación del Detector de Bordes de Canny sobre Redes Neuronales Celulares,» informe de investigación, Departamento de Electrónica e Ingeniería Electromecánica, Universidad de Extremadura, Badajoz, España. [En línea]. Available: https://www.researchgate.net/profile/Juan-Fernandez-Munoz/publication/228413045_Implementacion_del_detector_de_Bordes_de_Canny_sobre_Redes_Neuronales_Celulares/links/543ce5410cf2c432f74228b0/Implemen

tacion-del-detector-de-Bordes-de-Canny-sobre-Redes-Neuronales-Celulares.pdf. [Último acceso: Junio 2026].

[12] Y. Icer y M. Turk, «Implementation on Mainly Used Edge Detection Algorithms on FPGA,» *International Journal of Applied Mathematics, Electronics and Computers,* pp. 352-358, 2016.

[13] S. Vijayarani y M. Vinupriya, «Performance Analysis of Canny and Sobel Edge Detecction Algorithms in Image Mining,» *International Journal of Innovative Research in Computer and Communication Engineering,* vol. 1, 2013.

[14] A. Joy, J. Jacob y B. Roy, «High-Speed Hardware Edge Detection Implementation on FPGA using pipelined Sobel and Canny algorithms,» *Archives for Technical Sciences 2025,* vol. 33, nº 2, pp. 472-482, 2025.

[15] Python, «Python,» [En línea]. Available: https://docs.python.org/3/faq/general.html?.

[16] Python, «Wikipedia,» 2008. [En línea]. Available: https://es.wikipedia.org/wiki/Archivo:Python-logo-notext.svg.

[17] Microsoft, «Cose Visual Studio,» [En línea]. Available: https://code.visualstudio.com/docs/getstarted/getting-started. [Último acceso: Junio 2026].

[18] J. D. M. Frías, Introducción a los sistemas digitales, material docente de la asignatura, Escuela Superior Técnica de Ingeniería (ICAI), Universidad Pontificia Comillas, Madrid, España, 2023.

[19] AMD-Xinlix, «AMD products,» [En línea]. Available: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado.html?. [Último acceso: Junio 2026].

[20] AMD, «AMD vivado,» [En línea]. Available: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado/vivado-buy.html. [Último acceso: Junio 2026].

[21] T. Peng-o y P. Chaikan, «High performance and energy efficient sobel edge detection,» *Microprocessors and Microsystems,* vol. 87, Noviembre 2021.

[22] IEEE Standards Association, «numeric_std-body.vhdl,» IEEE Open Source Packages and Libraries, [En línea]. Available: https://opensource.ieee.org/vasg/Packages/-/blob/release/ieee/numeric_std-body.vhdl?. [Último acceso: Junio 2026].

[23] Digilent, «Reference Digilent,» [En línea]. Available: https://digilent.com/reference/basys3/refmanual. [Último acceso: Junio 2026].

[24] Digilent, «Shop products Digilent,» [En línea]. Available: https://digilent.com/shop/basys-3-amd-artix-7-fpga-trainer-board-recommended-for-introductory-users/?. [Último acceso: Junio 2026].

[25] Welectron, «Raspberry Pi 4 Modell B (4 GB RAM),» [En línea]. Available: https://www.welectron.com/Raspberry-Pi-4-Modell-B-4-GB-RAM. [Último acceso: Junio 2026].

[26] Glassdoor, «Sueldos de Software Engineer,» [En línea]. Available: https://www.glassdoor.es/Sueldos/software-engineer-sueldo-SRCH_KO0,17.htm. [Último acceso: Junio 2026].

[27] Talent, «Software developer: salario promedio en España, 2026,» [En línea]. Available: https://es.talent.com/salary?job=software+developer. [Último acceso: Junio 2026].

[28] Eurosat, «España-Precio de la electricidad en los hogares,» [En línea]. Available: https://datosmacro.expansion.com/energia-y-medio-ambiente/electricidad-precio-hogares/espana. [Último acceso: Junio 2026].

[29] Agencia Tributaria, «Manual de actividades económicas. Obligaciones fiscales de empresarios y profesionales residentes en territorio español,» [En línea]. Available: https://sede.agenciatributaria.gob.es/Sede/ayuda/manuales-videos-folletos/manuales-practicos/folleto-actividades-economicas/3-impuesto-sobre-renta-personas-fisicas/3_5-estimacion-directa-simplificada/3_5_4-tabla-amortizacion-simplificada.html. [Último acceso: Junio 2026].

[30] European Enviroment Agency, «El consumo europeo en una economía circular: las ventajas de unos dispositivos electrónicos más duraderos.,» [En línea]. Available: https://www.eea.europa.eu/en/analysis/publications/europes-consumption-in-a-circular-economy-the-benefits-of-longer-lasting-electronics. [Último acceso: Junio 2026].

[31] Naciones Unidas, «Objetivos de Desarrollo Sostenible,» [En línea]. Available: https://www.un.org/sustainabledevelopment/es/sustainable-development-goals/. [Último acceso: Marzo 2026].

[32] Correos, «Correos Market,» [En línea]. Available: https://www.market.correos.es/product/kingston-tarjeta-microsd-kingston-canvas-select-plus-almacenamiento-de-32?. [Último acceso: Junio 2026].

[33] Efecto2000, «Monitor táctil capacitivo 19" VGA HDMI USB,» [En línea]. Available: https://www.efecto2000.es/prod/monitores-tactiles/296942-iggual-monitor-tactil-capacitivo-19-vga-hdmi-usb-8435364319642.html?. [Último acceso: Junio 2026].

ANEXO I. ALINEACIÓN CON LOS ODS

El desarrollo de este Trabajo de Fin de Grado pertenece principalmente al ámbito de la innovación tecnológica, en concreto al área del procesamiento de imágenes. Aunque se trata de un trabajo de carácter técnico, sus resultados y formas de llevar a cabo el trabajo están alineadas y presentan una contribución a varios de los Objetivos de Desarrollo Sostenible (ODS) definidos por las Naciones Unidas [31]:

- **ODS 4: Educación de calidad**
  El presente proyecto se alinea con el ODS 4 al fortalecer la formación técnica en el ámbito de la ingeniería. Proporciona una vía de aprendizaje y fomenta la educación en el ámbito técnico.

- **ODS 9: Industria, Innovación e Infraestructuras**
  El trabajo se alinea directamente con el Objetivo de Desarrollo Sostenible 9, al tratarse de un trabajo que busca soluciones tecnológicas avanzadas. Este trabajo de innovación técnica busca actualizar y comprobar el funcionamiento de avances ya realizados en la industria, promoviendo así la inversión y el avance en desarrollo industrial y en el progreso tecnológico.

- **ODS 3: Salud y bienestar**
  El Objetivo de Desarrollo Sostenible 3, busca fomentar el bienestar universal, siendo fundamental garantizar una vida saludable de todas las personas. El trabajo se alinea con este objetivo debido a sus amplias aplicaciones en medicina.

- **ODS 7: Energía asequible y no contaminante**
  El Objetivo número 7 de la lista formulada por las Naciones Unidas propone una transición hacia un sistema energético asequible, seguro y sostenible, priorizando la implementación de prácticas eficientes. Este trabajo estudia la mejora de la eficiencia en una de las funciones más demandadas en el ámbito tecnológico.

- **ODS 10: Reducción de las desigualdades**
  El ODS 10 promueve la igualdad en todos los ámbitos y propone redoblar los esfuerzos para erradicar la pobreza y el hambre. En este caso, como el trabajo se centra en la propuesta de una solución tecnológica de bajo coste, que proporciona un mayor acceso a la tecnología, está contribuyendo indirectamente con este objetivo.

ANEXO II. CÓDIGO FUENTE

- **SOBEL:**

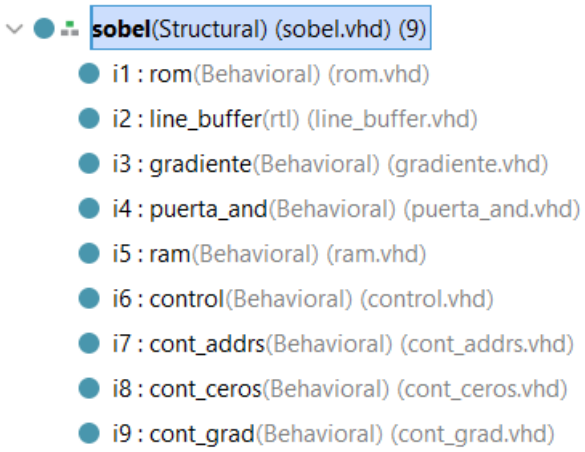


Figura 78. Jerarquía Sobel

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity rom is
    generic(
        WIDTH : integer:=640;
        HEIGHT : integer:=480
    );

    Port ( clk : in std_logic;
           reset: in std_logic;
           pix_in: in std_logic_vector (7 downto 0);
           enable: in std_logic;
           en_guardar: in std_logic;
           addrs : in integer range 0 to WIDTH*HEIGHT-1;
           pix_out : out std_logic_vector(7 downto 0);
           co_guardar: out std_logic);
end rom;

architecture Behavioral of rom is

    type rom_type is array (0 to WIDTH*HEIGHT-1) of std_logic_vector(7 downto 0);

    signal rom : rom_type;
    signal i: integer range 0 to WIDTH*HEIGHT-1;
```

```
begin

    process(clk, reset)
    begin
        if reset='1' then
            i<=0;
        elsif clk' event and clk='1' then
            if en_guardar='1' then
                if i<=(WIDTH*HEIGHT-1) then
                    rom(i)<=pix_in;
                    i<=i+1;
                else
                    i<=0;
                end if;
            end if;
            if enable='1' then
                pix_out<=rom(addrs);
                i<=0;
            end if;
        end if;
    end process;
    co_guardar<='1' when i=(WIDTH*HEIGHT-1) and en_guardar='1' else '0';
end Behavioral;
```

Código 1. rom.vhd en Sobel

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity line_buffer is
    generic (
        WIDTH : integer := 640
    );
    port(
        clk     : in  std_logic;
        reset   : in  std_logic;
        pix_in  : in  std_logic_vector(7 downto 0);
        p00, p01, p02,
        p10, p11, p12,
        p20, p21, p22 : out std_logic_vector(7 downto 0)
    );
end line_buffer;

architecture rtl of line_buffer is

    type t_row_mem is array (0 to WIDTH - 1) of std_logic_vector(7 downto 0);

    signal ram1 : t_row_mem;
    signal ram2 : t_row_mem;

    signal ptr : integer range 0 to WIDTH - 1 := 0;
```

```
    signal lb1_out : std_logic_vector(7 downto 0);
    signal lb2_out : std_logic_vector(7 downto 0);

    signal r : std_logic_vector(7 downto 0) := (others => '0');

    signal n_col1,  n_col0  : std_logic_vector(7 downto 0) := (others => '0');
    signal m_col1,  m_col0  : std_logic_vector(7 downto 0) := (others => '0');
    signal t_col1,  t_col0  : std_logic_vector(7 downto 0) := (others => '0');

begin

    lb1_out <= ram1(ptr);
    lb2_out <= ram2(ptr);

    process(clk, reset)
    begin
        if reset = '1' then
            ptr    <= 0;
            r      <= (others => '0');
            n_col1 <= (others => '0'); n_col0 <= (others => '0');
            m_col1 <= (others => '0'); m_col0 <= (others => '0');
            t_col1 <= (others => '0'); t_col0 <= (others => '0');

        elsif rising_edge(clk) then
            r <= pix_in;

            ram1(ptr) <= r;
            ram2(ptr) <= lb1_out;

            if ptr = WIDTH - 1 then
                ptr <= 0;
            else
                ptr <= ptr + 1;
            end if;

            n_col1 <= r;
            n_col0 <= n_col1;
            m_col1 <= lb1_out;
            m_col0 <= m_col1;
            t_col1 <= lb2_out;
            t_col0 <= t_col1;
        end if;
    end process;

    p22 <= r;        p21 <= n_col1;  p20 <= n_col0;   -- fila N
    p12 <= lb1_out;  p11 <= m_col1;  p10 <= m_col0;   -- fila N-1
    p02 <= lb2_out;  p01 <= t_col1;  p00 <= t_col0;   -- fila N-2

end architecture rtl;
```

Código 2. line_buffer.vhd en Sobel

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity gradiente is
    generic (THRESHOLD : integer := 500);
    Port ( clk : in std_logic;
           p00, p01, p02: in std_logic_vector (7 downto 0);
           p10, p11, p12: in std_logic_vector (7 downto 0);
           p20, p21, p22: in std_logic_vector (7 downto 0);
           borde: out std_logic);
end gradiente;

architecture Behavioral of gradiente is
      signal Gx, Gy: signed (10 downto 0);
      signal G: unsigned (11 downto 0);
begin

    Calculo: process (clk)
    begin
       if clk'event and clk= '1' then
           Gx<=
                -signed(resize(unsigned(p00), 11))
                +signed(resize(unsigned(p02), 11))
                - shift_left(signed(resize(unsigned(p10), 11)),1)
                +shift_left(signed(resize(unsigned(p12), 11)),1)
                - signed(resize(unsigned(p20), 11))
                +signed(resize(unsigned(p22), 11));
           Gy<=
                signed(resize(unsigned(p00), 11))
                -signed(resize(unsigned(p20), 11))
                +shift_left(signed(resize(unsigned(p01), 11)),1)
                -shift_left(signed(resize(unsigned(p21), 11)),1)
                +signed(resize(unsigned(p02), 11))
                -signed(resize(unsigned(p22), 11));
       end if;
    end process;

    G<=resize(unsigned(abs(Gx)),12)+resize(unsigned(abs(Gy)),12);
    borde<='1' when (G> THRESHOLD) else '0';
end Behavioral;
```

Código 3. gradiente.vhd en Sobel

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity puerta_and is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           s : out STD_LOGIC);
end puerta_and;

architecture Behavioral of puerta_and is
```

```
begin

    s<= a and b;

end Behavioral;
```

Código 4. puerta_and.vhd en Sobel

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ram is
    generic(
        WIDTH : integer:=10;
        HEIGHT : integer:=10
    );
    Port ( clk : in STD_LOGIC;
           enable : in STD_LOGIC;
           reset : in STD_LOGIC;
           pix_in: in std_logic;
           rd_addr : in  integer range 0 to WIDTH*HEIGHT-1;
           rd_data : out std_logic);
end ram;

architecture Behavioral of ram is
    type ram_type is array (0 to WIDTH*HEIGHT-1) of std_logic;
    signal ram: ram_type;
    signal i: integer range 0 to WIDTH*HEIGHT-1;
begin
    process (clk, reset)
    begin
        if reset = '1' then
            i <= 0;                          --no es necesario limpiar la memoria
en el reset
        elsif clk' event and clk='1' then
            if enable='1' then
                if i<(WIDTH*HEIGHT-1) then
                    ram(i)<=pix_in;
                    i<=i+1;
                else
                    i<=0;
                end if;
            end if;
```

```
        end if;
    end process;

    process(clk)
    begin
        if rising_edge(clk) then
            rd_data <= ram(rd_addr);
        end if;
    end process;

end Behavioral;
```

Código 5. ram.vhd en Sobel

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity control is
    generic( WIDTH: integer := 640;
             HEIGHT: integer := 480);

    Port ( clk : in std_logic;
           reset: in std_logic;
           puls: in std_logic;
           addrs: in integer range 0 to WIDTH*HEIGHT-1;
           fin_espera: in std_logic;
           co_addrs: in std_logic;
           co_ceros: in std_logic;
           co_grad: in std_logic;
           co_guardar: in std_logic;
           en_rom: out std_logic;
           en_ram: out std_logic;
           guardarcero: out std_logic;
           en_addrs: out std_logic;
           en_ceros: out std_logic;
           en_grad: out std_logic;
           en_guardar: out std_logic;
           imagen_ok: out std_logic;
           reset_addrs: out std_logic;
           reset_ceros: out std_logic;
           reset_grad: out std_logic;
           reset_ram: out std_logic;
           reset_rom: out std_logic;
           reset_buffer: out std_logic
           );
end control;

architecture Behavioral of control is
    type t_estados is (Reposo, Pulso, Espera, Guardar, NoGuardar, Ceros,
Gradiente, Izda, Dcha, Ultima, Fin);
    signal estado_act, estado_sig: t_estados;
begin
    VarEstado: process (clk, reset)
```

```
    begin
        if reset = '1' then
            estado_act<=Reposo;
        elsif clk' event and clk='1' then
            estado_act<=estado_sig;
        end if;
    end process VarEstado;

    TransicionEstados: process (estado_act, puls, fin_espera, addrs, co_addrs,
co_ceros, co_grad, co_guardar)
    begin
        estado_sig<=estado_act;
        case estado_act is
            when Reposo =>
                if puls='1' then
                    estado_sig<=Pulso;
                end if;

            when Pulso =>
                estado_sig<=Espera;

            when Espera =>
                if fin_espera='1' then
                    estado_sig<=Guardar;
                end if;

            when Guardar =>
                if co_guardar='1' then
                    estado_sig<=NoGuardar;
                end if;

            when NoGuardar =>
                if addrs>=(WIDTH+4) then
                    estado_sig<=Ceros;
                end if;

            when Ceros =>
                if co_ceros='1' then
                    estado_sig<=Izda;
                end if;

            when Izda=>
                estado_sig<=Gradiente;

            when Dcha=>
                if co_addrs='1' then
                    estado_sig<=Ultima;
                else
                    estado_sig<=Izda;
                end if;

            when Gradiente=>
```

```
            if co_grad='1' then
                estado_sig<=Dcha;
            end if;

        when Ultima=>
            if co_ceros='1' then
                estado_sig<=Fin;
            end if;

        when Fin=>
            estado_sig<=Pulso;
        end case;
    end process TransicionEstados;

    Salidas: process (estado_act)
    begin
        case estado_act is
            when Reposo=>
                imagen_ok<='0';
                en_guardar<='0';
                en_addrs<='0';
                en_ceros<='0';
                en_grad<='0';
                reset_rom<='1';
                reset_ram<='1';
                reset_addrs<='1';
                reset_ceros<='1';
                reset_grad<='1';
                reset_buffer<='1';
                en_rom<='0';
                en_ram<='0';
                guardarcero<='0';


            when Pulso=>
                imagen_ok<='1';
                en_guardar<='0';
                en_addrs<='0';
                en_ceros<='0';
                en_grad<='0';
                reset_rom<='1';
                reset_ram<='1';
                reset_addrs<='1';
                reset_ceros<='1';
                reset_grad<='1';
                reset_buffer<='1';
                en_rom<='0';
                en_ram<='0';
                guardarcero<='0';


            when Espera=>
```

```
                imagen_ok<='0';
                en_guardar<='0';
                en_addrs<='0';
                en_ceros<='0';
                en_grad<='0';
                reset_rom<='1';
                reset_ram<='1';
                reset_addrs<='1';
                reset_ceros<='1';
                reset_grad<='1';
                reset_buffer<='1';
                en_rom<='0';
                en_ram<='0';
                guardarcero<='0';


            when Guardar=>
                imagen_ok<='0';
                en_guardar<='1';
                en_addrs<='0';
                en_ceros<='0';
                en_grad<='0';
                reset_rom<='0';
                reset_ram<='1';
                reset_addrs<='1';
                reset_ceros<='1';
                reset_grad<='1';
                reset_buffer<='1';
                en_rom<='0';
                en_ram<='0';
                guardarcero<='0';


            when NoGuardar=>
                imagen_ok<='0';
                en_guardar<='0';
                en_addrs<='1';
                en_ceros<='0';
                en_grad<='0';
                reset_rom<='0';
                reset_ram<='1';
                reset_addrs<='0';
                reset_ceros<='1';
                reset_grad<='1';
                reset_buffer<='0';
                en_rom<='1';
                en_ram<='0';
                guardarcero<='0';


            when Ceros=>
                imagen_ok<='0';
```

```
                    en_guardar<='0';
                    en_addrs<='1';
                    en_ceros<='1';
                    en_grad<='0';
                    reset_rom<='0';
                    reset_ram<='0';
                    reset_addrs<='0';
                    reset_ceros<='0';
                    reset_grad<='1';
                    reset_buffer<='0';
                    en_rom<='1';
                    en_ram<='1';
                    guardarcero<='0';


                when Izda=>
                    imagen_ok<='0';
                    en_guardar<='0';
                    en_addrs<='1';
                    en_ceros<='0';
                    en_grad<='0';
                    reset_rom<='0';
                    reset_ram<='0';
                    reset_addrs<='0';
                    reset_ceros<='1';
                    reset_grad<='1';
                    reset_buffer<='0';
                    en_rom<='1';
                    en_ram<='1';
                    guardarcero<='0';


                when Dcha=>
                    imagen_ok<='0';
                    en_guardar<='0';
                    en_addrs<='1';
                    en_ceros<='0';
                    en_grad<='0';
                    reset_rom<='0';
                    reset_ram<='0';
                    reset_addrs<='0';
                    reset_ceros<='1';
                    reset_grad<='1';
                    reset_buffer<='0';
                    en_rom<='1';
                    en_ram<='1';
                    guardarcero<='0';


                when Gradiente=>
                    imagen_ok<='0';
                    en_guardar<='0';
```

```
                    en_addrs<='1';
                    en_ceros<='0';
                    en_grad<='1';
                    reset_rom<='0';
                    reset_ram<='0';
                    reset_addrs<='0';
                    reset_ceros<='1';
                    reset_grad<='0';
                    reset_buffer<='0';
                    en_rom<='1';
                    en_ram<='1';
                    guardarcero<='1';


                when Ultima=>
                    imagen_ok<='0';
                    en_guardar<='0';
                    en_addrs<='1';
                    en_ceros<='1';
                    en_grad<='0';
                    reset_rom<='0';
                    reset_ram<='0';
                    reset_addrs<='0';
                    reset_ceros<='0';
                    reset_grad<='1';
                    reset_buffer<='0';
                    en_rom<='1';
                    en_ram<='1';
                    guardarcero<='0';


                when Fin=>
                    imagen_ok<='0';
                    en_guardar<='0';
                    en_addrs<='0';
                    en_ceros<='0';
                    en_grad<='0';
                    reset_rom<='1';
                    reset_ram<='1';
                    reset_addrs<='1';
                    reset_ceros<='1';
                    reset_grad<='1';
                    reset_buffer<='1';
                    en_rom<='0';
                    en_ram<='0';
                    guardarcero<='0';


                end case;
            end process Salidas;

end Behavioral;
```

Código 6. control.vhd en Sobel

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity cont_addrs is
    generic(WIDTH: integer := 640;
            HEIGHT: integer := 480);

    Port ( clk : in std_logic;
           reset: in std_logic;
           enable : in std_logic;
           co : out std_logic;
           s: out integer range 0 to WIDTH*HEIGHT-1);
end cont_addrs;

architecture Behavioral of cont_addrs is
    signal contador: integer range 0 to WIDTH*HEIGHT-1;
begin
    process (clk, reset)
    begin
        if reset = '1' then
            contador<= 0;
        elsif clk' event and clk ='1' then
            if enable = '1' then
                if contador<(HEIGHT*WIDTH-1) then
                    contador<= contador+1;
                end if;
            end if;
        end if;
    end process;

    co<='1' when contador =(WIDTH*HEIGHT-1) else '0';
    s <= contador;

end Behavioral;
```

Código 7. cont_addrs.vhd en Sobel

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity cont_ceros is
    generic(WIDTH: integer := 640);

    Port ( clk : in STD_LOGIC;
           reset: in std_logic;
           enable : in STD_LOGIC;
           co : out STD_LOGIC);
end cont_ceros;

architecture Behavioral of cont_ceros is
    signal contador: integer range 0 to WIDTH-1;
```

```
begin
    process (clk, reset)
    begin
        if reset = '1' then
            contador<= 0;
        elsif clk' event and clk ='1' then
            if enable = '1' then
                if contador=(WIDTH-1) then
                    contador <= 0;
                else
                    contador<= contador+1;
                end if;
            end if;
        end if;
    end process;

    co<='1' when contador =(WIDTH-1) and enable='1' else '0';


end Behavioral;
```

Código 8. cont_ceros.vhd en Sobel

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity cont_grad is
    generic(WIDTH: integer := 640);

    Port ( clk : in STD_LOGIC;
           reset: in std_logic;
           enable : in STD_LOGIC;
           co : out STD_LOGIC);
end cont_grad;

architecture Behavioral of cont_grad is
    signal contador: integer range 0 to WIDTH-3;
begin
    process (clk, reset)
    begin
        if reset = '1' then
            contador<= 0;
        elsif clk' event and clk ='1' then
            if enable = '1' then
                if contador=(WIDTH-3) then
                    contador <= 0;
                else
                    contador<= contador+1;
                end if;
            end if;
        end if;
    end process;
```

```
    co<='1' when contador =(WIDTH-3) and enable='1' else '0';


end Behavioral;
```

Código 9. cont_grad.vhd en Sobel

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sobel is
    generic(
        WIDTH : integer:=640;
        HEIGHT : integer:=480;
        THRESHOLD : integer:=500
    );
    Port ( clk : in STD_LOGIC;
           reset: in std_logic;
           puls: in std_logic;
           pix_in: in std_logic_vector (7 downto 0);
           fin_espera: in std_logic;
           rd_addr: in  integer range 0 to WIDTH*HEIGHT-1;
           en_camara: out std_logic;
           rd_data : out std_logic
           );
end sobel;

architecture Structural of sobel is


    component rom
        generic(
        WIDTH : integer:=640;
        HEIGHT : integer:=480
        );
        Port ( clk : in STD_LOGIC;
               reset: in std_logic;
               pix_in: in std_logic_vector (7 downto 0);
               enable: in std_logic;
               en_guardar: in std_logic;
               addrs : in integer range 0 to WIDTH*HEIGHT-1;
               pix_out : out std_logic_vector(7 downto 0);
               co_guardar: out std_logic);
   end component;

   component line_buffer
        generic (
            WIDTH : integer := 640
        );
        port( clk: in std_logic;
            reset: in std_logic;
            pix_in: in std_logic_vector(7 downto 0);
```

```
            p00,p01,p02, p10, p11, p12, p20, p21, p22: out std_logic_vector(7
downto 0));
   end component;

   component gradiente
        generic (THRESHOLD : integer := 500);
        port ( clk : in std_logic;
           p00, p01, p02: in std_logic_vector (7 downto 0);
           p10, p11, p12: in std_logic_vector (7 downto 0);
           p20, p21, p22: in std_logic_vector (7 downto 0);
           borde: out std_logic);
   end component;

   component puerta_and
        Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           s : out STD_LOGIC);
   end component;

   component ram
        generic(
        WIDTH : integer:=640;
        HEIGHT : integer:=480
    );
    Port ( clk : in STD_LOGIC;
           enable : in STD_LOGIC;
           reset : in STD_LOGIC;
           pix_in: in std_logic;
           rd_addr : in  integer range 0 to WIDTH*HEIGHT-1;
           rd_data : out std_logic);
   end component;

   component control
        generic( WIDTH: integer := 640;
             HEIGHT: integer := 480);

    Port ( clk : in std_logic;
           reset: in std_logic;
           puls: in std_logic;
           addrs: in integer range 0 to WIDTH*HEIGHT-1;
           fin_espera: in std_logic;
           co_addrs: in std_logic;
           co_ceros: in std_logic;
           co_grad: in std_logic;
           co_guardar: in std_logic;
           en_rom: out std_logic;
           en_ram: out std_logic;
           guardarcero: out std_logic;
           en_addrs: out std_logic;
           en_ceros: out std_logic;
           en_grad: out std_logic;
           en_guardar: out std_logic;
```

```
           imagen_ok: out std_logic;
           reset_addrs: out std_logic;
           reset_ceros: out std_logic;
           reset_grad: out std_logic;
           reset_ram: out std_logic;
           reset_rom: out std_logic;
           reset_buffer: out std_logic
           );
   end component;

   component cont_addrs
        generic(WIDTH: integer := 640;
            HEIGHT: integer := 480);
        Port ( clk : in std_logic;
           reset: in std_logic;
           enable : in std_logic;
           co : out std_logic;
           s: out integer range 0 to WIDTH*HEIGHT-1);
   end component;

   component cont_ceros
        generic(WIDTH: integer := 640);
        Port ( clk : in STD_LOGIC;
           reset: in std_logic;
           enable : in STD_LOGIC;
           co : out STD_LOGIC);
   end component;

   component cont_grad
        generic(WIDTH: integer := 640);
        Port ( clk : in STD_LOGIC;
           reset: in std_logic;
           enable : in STD_LOGIC;
           co : out STD_LOGIC);
   end component;

    signal out_rom: std_logic_vector(7 downto 0);
    signal p00, p01, p02, p10, p11, p12, p20, p21, p22: std_logic_vector (7
downto 0);
    signal out_grad, in_ram: std_logic;
    signal addrs: integer range 0 to WIDTH*HEIGHT-1;
    signal guardarceros: std_logic;
    signal en_rom, en_ram, en_addrs, en_ceros, en_grad, en_guardar: std_logic;
    signal reset_linebuffer, reset_ram, reset_addrs, reset_ceros, reset_grad,
reset_rom: std_logic;
    signal co_addrs, co_ceros, co_grad, co_guardar: std_logic;

begin

    i1: rom
        generic map(
            WIDTH=>WIDTH,
```

```
            HEIGHT=>HEIGHT)
        port map(
            clk=>clk,
            reset=>reset_rom,
            pix_in=>pix_in,
            enable=>en_rom,
            en_guardar=>en_guardar,
            addrs=>addrs,
            pix_out=>out_rom,
            co_guardar=>co_guardar);

    i2: line_buffer
        generic map(
            WIDTH=>WIDTH)
        port map(
            clk=>clk,
            reset=>reset_linebuffer,
            pix_in=>out_rom,
            p00=>p00,
            p01=>p01,
            p02=>p02,
            p10=>p10,
            p11=>p11,
            p12=>p12,
            p20=>p20,
            p21=>p21,
            p22=>p22);

    i3: gradiente
        generic map(
            THRESHOLD=>THRESHOLD)
        port map(
            clk=>clk,
            p00=>p00,
            p01=>p01,
            p02=>p02,
            p10=>p10,
            p11=>p11,
            p12=>p12,
            p20=>p20,
            p21=>p21,
            p22=>p22,
            borde=>out_grad);

    i4: puerta_and
        port map(
            a=>out_grad,
            b=>guardarceros,
            s=>in_ram);

    i5: ram
        generic map(
```

```
            WIDTH=>WIDTH,
            HEIGHT=>HEIGHT)
        port map(
            clk=>clk,
            enable=>en_ram,
            reset=>reset_ram,
            pix_in=>in_ram,
            rd_addr=>rd_addr,
            rd_data=>rd_data);

    i6: control
        generic map(
            WIDTH=>WIDTH,
            HEIGHT=>HEIGHT)
        port map(
            clk=>clk,
            reset=>reset,
            puls=>puls,
            addrs=>addrs,
            fin_espera=>fin_espera,
            co_addrs=>co_addrs,
            co_ceros=>co_ceros,
            co_grad=>co_grad,
            co_guardar=>co_guardar,
            en_rom=>en_rom,
            en_ram=>en_ram,
            guardarcero=>guardarceros,
            en_addrs=>en_addrs,
            en_ceros=>en_ceros,
            en_grad=>en_grad,
            en_guardar=>en_guardar,
            imagen_ok=>en_camara,
            reset_addrs=>reset_addrs,
            reset_ceros=>reset_ceros,
            reset_grad=>reset_grad,
            reset_rom=>reset_rom,
            reset_ram=>reset_ram,
            reset_buffer=>reset_linebuffer);

    i7: cont_addrs
            generic map(
                WIDTH=>WIDTH,
                HEIGHT=>HEIGHT)
            port map(
                clk=>clk,
                reset=>reset_addrs,
                enable=>en_addrs,
                co=>co_addrs,
                s=>addrs);
    i8: cont_ceros
            generic map(
                WIDTH=>WIDTH)
```

```
            port map(
                clk=>clk,
                reset=>reset_ceros,
                enable=>en_ceros,
                co=>co_ceros);

    i9: cont_grad
            generic map(
                WIDTH=>WIDTH)
            port map(
                clk=>clk,
                reset=>reset_grad,
                enable=>en_grad,
                co=>co_grad);

end Structural;
```

Código 10. sobel.vhd en Sobel

- **CANNY**

Figura 79. Jerarquía Canny

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity rom is
    generic(
        WIDTH : integer:=640;
        HEIGHT : integer:=480
    );

    Port ( clk : in std_logic;
           reset: in std_logic;
           pix_in: in std_logic_vector (7 downto 0);
           enable: in std_logic;
           en_guardar: in std_logic;
           addrs : in integer range 0 to WIDTH*HEIGHT-1;
           pix_out : out std_logic_vector(7 downto 0);
           co_guardar: out std_logic);
end rom;
```

```
architecture Behavioral of rom is

    type rom_type is array (0 to WIDTH*HEIGHT-1) of std_logic_vector(7 downto 0);

    signal rom : rom_type;
    signal i: integer range 0 to WIDTH*HEIGHT-1;

begin

    process(clk, reset)
    begin
        if reset='1' then
            i<=0;
        elsif clk' event and clk='1' then
            if en_guardar='1' then
                if i<=(WIDTH*HEIGHT-1) then
                    rom(i)<=pix_in;
                    i<=i+1;
                else
                    i<=0;
                end if;
            end if;
            if enable='1' then
                pix_out<=rom(addrs);
                i<=0;
            end if;
        end if;
    end process;

    co_guardar<='1' when i=(WIDTH*HEIGHT-1) and en_guardar='1' else '0';

end Behavioral;
```

Código 11. rom.vhd en Canny

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity line_buffer is
    generic (
        WIDTH : integer := 640
    );
    port(
        clk     : in  std_logic;
        reset   : in  std_logic;
        pix_in  : in  std_logic_vector(7 downto 0);
        p00, p01, p02,
        p10, p11, p12,
        p20, p21, p22 : out std_logic_vector(7 downto 0)
    );
end line_buffer;
```

```
architecture rtl of line_buffer is

    type t_row_mem is array (0 to WIDTH - 1) of std_logic_vector(7 downto 0);

    signal ram1 : t_row_mem;
    signal ram2 : t_row_mem;

    signal ptr : integer range 0 to WIDTH - 1 := 0;

    signal lb1_out : std_logic_vector(7 downto 0);
    signal lb2_out : std_logic_vector(7 downto 0);

    signal r : std_logic_vector(7 downto 0) := (others => '0');

    signal n_col1,  n_col0  : std_logic_vector(7 downto 0) := (others => '0');
    signal m_col1,  m_col0  : std_logic_vector(7 downto 0) := (others => '0');
    signal t_col1,  t_col0  : std_logic_vector(7 downto 0) := (others => '0');

begin

    lb1_out <= ram1(ptr);
    lb2_out <= ram2(ptr);

    process(clk, reset)
    begin
        if reset = '1' then
            ptr    <= 0;
            r      <= (others => '0');
            n_col1 <= (others => '0'); n_col0 <= (others => '0');
            m_col1 <= (others => '0'); m_col0 <= (others => '0');
            t_col1 <= (others => '0'); t_col0 <= (others => '0');

        elsif rising_edge(clk) then
            r <= pix_in;

            ram1(ptr) <= r;
            ram2(ptr) <= lb1_out;

            if ptr = WIDTH - 1 then
                ptr <= 0;
            else
                ptr <= ptr + 1;
            end if;

            n_col1 <= r;
            n_col0 <= n_col1;
            m_col1 <= lb1_out;
            m_col0 <= m_col1;
            t_col1 <= lb2_out;
            t_col0 <= t_col1;
        end if;
    end process;
```

```
    p22 <= r;        p21 <= n_col1;  p20 <= n_col0;   -- fila N
    p12 <= lb1_out;  p11 <= m_col1;  p10 <= m_col0;   -- fila N-1
    p02 <= lb2_out;  p01 <= t_col1;  p00 <= t_col0;   -- fila N-2

end architecture rtl;
```

Código 12. line_buffer.vhd en Canny

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity suavizado is
    generic(
        WIDTH : integer:=640;
        HEIGHT : integer:=480
    );
    Port ( clk : in std_logic;
           p00, p01, p02: in std_logic_vector (7 downto 0);
           p10, p11, p12: in std_logic_vector (7 downto 0);
           p20, p21, p22:in std_logic_vector (7 downto 0);
           pix_out : out std_logic_vector (7 downto 0));
end suavizado;

architecture Behavioral of suavizado is

    signal intensidad: unsigned (7 downto 0);

begin
    Calculo: process (clk)
    begin
        if clk'event and clk= '1' then
            intensidad<=shift_right(unsigned(p00),4)
                        +shift_right(unsigned(p01),3)
                        +shift_right(unsigned(p02),4)                -- 2/16=1/8
                        +shift_right(unsigned(p10),3)
                        +shift_right(unsigned(p11),2)
                        +shift_right(unsigned(p12),3)                -- 4/16=1/4
                        +shift_right(unsigned(p20),4)
                        +shift_right(unsigned(p21),3)
                        +shift_right(unsigned(p22),4);
        end if;
    end process;

    pix_out<=std_logic_vector(intensidad);


end Behavioral;
```

Código 13. suavizado.vhd en Canny

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Mux is
    Port ( a : in STD_LOGIC_vector (7 downto 0);
           b : in STD_LOGIC_vector (7 downto 0);
           sel : in STD_LOGIC;
           s : out STD_LOGIC_vector (7 downto 0));
end Mux;

architecture Behavioral of Mux is

begin

    with sel select
        s<= a when '0',
            b when others;

end Behavioral;
```

Código 14. Mux.vhd en Canny

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity gradiente is
    Port ( clk : in std_logic;
           p00, p01, p02: in std_logic_vector (7 downto 0);
           p10, p11, p12: in std_logic_vector (7 downto 0);
           p20, p21, p22: in std_logic_vector (7 downto 0);
           grad : out std_logic_vector (11 downto 0);
           angulo : out std_logic_vector (1 downto 0));
end gradiente;

architecture Behavioral of gradiente is
    signal Gx, Gy: signed (10 downto 0);
    signal abs_gx, abs_gy: signed (11 downto 0);
    signal G: unsigned (11 downto 0);
begin

    Calculo: process (clk)
    begin
        if clk'event and clk= '1' then
                Gx<=
                -signed(resize(unsigned(p00), 11))
                +signed(resize(unsigned(p02), 11))
                -shift_left(signed(resize(unsigned(p10), 11)),1)
                +shift_left(signed(resize(unsigned(p12), 11)),1)
                -signed(resize(unsigned(p20), 11))
                +signed(resize(unsigned(p22), 11));

                Gy<=
                signed(resize(unsigned(p00), 11))
```

```
                -signed(resize(unsigned(p20), 11))
                +shift_left(signed(resize(unsigned(p01), 11)),1)
                -shift_left(signed(resize(unsigned(p21), 11)),1)
                +signed(resize(unsigned(p02), 11))
                -signed(resize(unsigned(p22), 11));
         end if;
    end process;

    abs_gx<=resize(abs(Gx),12);
    abs_gy<=resize(abs(Gy),12);
    G<=resize(unsigned(abs_gx),12)+resize(unsigned(abs_gy),12);

    grad<=std_logic_vector(G);

    angulo<= "00" when (abs_gx>shift_left(abs_gy,1)) else
             "10" when (abs_gy>shift_left(abs_gx,1)) else
             "01" when (Gx>0 and Gy>0) or (Gx<0 and Gy<0) else
             "11";

end Behavioral;
```

Código 15. gradiente.vhd en Canny

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity line_buffer11 is
    generic (
        WIDTH : integer := 640
    );
    port(
        clk     : in  std_logic;
        reset   : in  std_logic;
        pix_in  : in  std_logic_vector(11 downto 0);
        p00, p01, p02,
        p10, p11, p12,
        p20, p21, p22 : out std_logic_vector(11 downto 0)
    );
end line_buffer11;

architecture rtl of line_buffer11 is

    type t_row_mem is array (0 to WIDTH - 1) of std_logic_vector(11 downto 0);

    signal ram1 : t_row_mem;
    signal ram2 : t_row_mem;

    signal ptr : integer range 0 to WIDTH - 1 := 0;

    signal lb1_out : std_logic_vector(11 downto 0);
    signal lb2_out : std_logic_vector(11 downto 0);
```

```
    signal r : std_logic_vector(11 downto 0) := (others => '0');

    signal n_col1,  n_col0  : std_logic_vector(11 downto 0) := (others => '0');
    signal m_col1,  m_col0  : std_logic_vector(11 downto 0) := (others => '0');
    signal t_col1,  t_col0  : std_logic_vector(11 downto 0) := (others => '0');

begin

    lb1_out <= ram1(ptr);
    lb2_out <= ram2(ptr);

    process(clk, reset)
    begin
        if reset = '1' then
            ptr    <= 0;
            r      <= (others => '0');
            n_col1 <= (others => '0'); n_col0 <= (others => '0');
            m_col1 <= (others => '0'); m_col0 <= (others => '0');
            t_col1 <= (others => '0'); t_col0 <= (others => '0');

        elsif rising_edge(clk) then
            r <= pix_in;

            ram1(ptr) <= r;
            ram2(ptr) <= lb1_out;

            if ptr = WIDTH - 1 then
                ptr <= 0;
            else
                ptr <= ptr + 1;
            end if;

            n_col1 <= r;
            n_col0 <= n_col1;
            m_col1 <= lb1_out;
            m_col0 <= m_col1;
            t_col1 <= lb2_out;
            t_col0 <= t_col1;
        end if;
    end process;

    p22 <= r;        p21 <= n_col1;  p20 <= n_col0;   -- fila N
    p12 <= lb1_out;  p11 <= m_col1;  p10 <= m_col0;   -- fila N-1
    p02 <= lb2_out;  p01 <= t_col1;  p00 <= t_col0;   -- fila N-2

end architecture rtl;
```

Código 16. line_buffer11.vhd en Canny

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
```

```
entity line_buffer1 is
    generic (
        WIDTH : integer := 640
    );
    port(
        clk     : in  std_logic;
        reset   : in  std_logic;
        pix_in  : in  std_logic_vector(1 downto 0);
        p00, p01, p02,
        p10, p11, p12,
        p20, p21, p22 : out std_logic_vector(1 downto 0)
    );
end line_buffer1;

architecture rtl of line_buffer1 is

    type t_row_mem is array (0 to WIDTH - 1) of std_logic_vector(1 downto 0);

    signal ram1 : t_row_mem;
    signal ram2 : t_row_mem;

    signal ptr : integer range 0 to WIDTH - 1 := 0;

    signal lb1_out : std_logic_vector(1 downto 0);
    signal lb2_out : std_logic_vector(1 downto 0);

    signal r : std_logic_vector(1 downto 0) := (others => '0');

    signal n_col1,  n_col0  : std_logic_vector(1 downto 0) := (others => '0');
    signal m_col1,  m_col0  : std_logic_vector(1 downto 0) := (others => '0');
    signal t_col1,  t_col0  : std_logic_vector(1 downto 0) := (others => '0');

begin


    lb1_out <= ram1(ptr);
    lb2_out <= ram2(ptr);

    process(clk, reset)
    begin
        if reset = '1' then
            ptr    <= 0;
            r      <= (others => '0');
            n_col1 <= (others => '0'); n_col0 <= (others => '0');
            m_col1 <= (others => '0'); m_col0 <= (others => '0');
            t_col1 <= (others => '0'); t_col0 <= (others => '0');

        elsif rising_edge(clk) then
            r <= pix_in;

            ram1(ptr) <= r;
            ram2(ptr) <= lb1_out;
```

```
            if ptr = WIDTH - 1 then
                ptr <= 0;
            else
                ptr <= ptr + 1;
            end if;

            n_col1 <= r;
            n_col0 <= n_col1;
            m_col1 <= lb1_out;
            m_col0 <= m_col1;
            t_col1 <= lb2_out;
            t_col0 <= t_col1;
        end if;
    end process;

    p22 <= r;        p21 <= n_col1;  p20 <= n_col0;   -- fila N
    p12 <= lb1_out;  p11 <= m_col1;  p10 <= m_col0;   -- fila N-1
    p02 <= lb2_out;  p01 <= t_col1;  p00 <= t_col0;   -- fila N-2

end architecture rtl;
```

Código 17. line_buffer1.vhd en Canny

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity supresion is
    generic (Talto : integer :=200;
             Tbajo: integer :=80);

    Port ( clk : in std_logic;
           g00, g01, g02: in std_logic_vector (11 downto 0);
           g10, g11, g12: in std_logic_vector (11 downto 0);
           g20, g21, g22: in std_logic_vector (11 downto 0);
           a00, a01, a02: in std_logic_vector (1 downto 0);
           a10, a11, a12: in std_logic_vector (1 downto 0);
           a20, a21, a22: in std_logic_vector (1 downto 0);
           tipoborde: out std_logic_vector (1 downto 0));
end supresion;

architecture Behavioral of supresion is

begin

    calculo: process (clk)
    begin
    if clk'event and clk='1' then
        if (a11="00") then
            if unsigned(g12)>unsigned(g11) or unsigned(g10)>unsigned(g11) or
unsigned(g11)<Tbajo then
                tipoborde<="00";                --no borde
```

```
            elsif unsigned(g11)>Talto then
                tipoborde<="10";                --borde fuerte
            else
                tipoborde<="01";                --borde debil
            end if;
        end if;

        if (a11="01") then
            if unsigned(g02)>unsigned(g11) or unsigned(g20)>unsigned(g11) or
unsigned(g11)<Tbajo then
                tipoborde<="00";                --no borde
            elsif unsigned(g11)>Talto then
                tipoborde<="10";                --borde fuerte
            else
                tipoborde<="01";                --borde debil
            end if;
        end if;

        if (a11="10") then
            if unsigned(g01)>unsigned(g11) or unsigned(g21)>unsigned(g11) or
unsigned(g11)<Tbajo then
                tipoborde<="00";                --no borde
            elsif unsigned(g11)>Talto then
                tipoborde<="10";                --borde fuerte
            else
                tipoborde<="01";                --borde debil
            end if;
        end if;

        if (a11="11") then
            if unsigned(g00)>unsigned(g11) or unsigned(g22)>unsigned(g11) or
unsigned(g11)<Tbajo then
                tipoborde<="00";                --no borde
            elsif unsigned(g11)>Talto then
                tipoborde<="10";                --borde fuerte
            else
                tipoborde<="01";                --borde debil
            end if;
        end if;
    end if;
    end process;

end Behavioral;
```

Código 18. supresion.vhd en Canny

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity histeresis is
    Port ( clk : in std_logic;
           t00, t01, t02: in std_logic_vector (1 downto 0);
           t10, t11, t12: in std_logic_vector (1 downto 0);
           t20, t21, t22: in std_logic_vector (1 downto 0);
```

```
           borde : out std_logic);
end histeresis;

architecture Behavioral of histeresis is

begin

    calculo: process (clk)
    begin
        if clk'event and clk='1' then
            if t11="00" then
                borde<='0';
            elsif t11="10" then
                borde<='1';
            elsif t11="01" then
                if t00="10" or t01="10" or t02="10" or t10="10"
                or t12="10" or t20="10" or t21="10" or t22="10" then
                    borde<='1';
                else
                    borde<='0';
                end if;
            end if;
        end if;
    end process;
end Behavioral;
```

Código 19. histeresis.vhd en Canny

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity puerta_and is
    Port ( a : in std_logic;
           b : in std_logic;
           s : out std_logic);
end puerta_and;

architecture Behavioral of puerta_and is

begin

    s<= a and b;

end Behavioral;
```

Código 20. puerta_and.vhd en Canny

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ram is
    generic(
        WIDTH : integer:=640;
        HEIGHT : integer:=480
    );
```

```
    Port ( clk : in std_logic;
           enable : in std_logic;
           reset : in std_logic;
           pix_in: in std_logic;
           rd_addr : in  integer range 0 to WIDTH*HEIGHT-1;
           rd_data : out std_logic);
end ram;

architecture Behavioral of ram is
    type ram_type is array (0 to WIDTH*HEIGHT-1) of std_logic;
    signal ram: ram_type;
    signal i: integer range 0 to WIDTH*HEIGHT-1;
begin

    process (clk, reset)
    begin
        if reset = '1' then
            i <= 0;
        elsif clk' event and clk='1' then
            if enable='1' then
                if i<(WIDTH*HEIGHT-1) then
                    ram(i)<=pix_in;
                    i<=i+1;
                else
                    i<=0;
                end if;
            end if;
        end if;
    end process;

    process(clk)
    begin
        if rising_edge(clk) then
            rd_data <= ram(rd_addr);
        end if;
    end process;

end behavioral;
```

Código 21. ram.vhd en Canny

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity control is
    generic(WIDTH: integer := 640;
            HEIGHT: integer := 480);
    Port ( clk : in std_logic;
           reset : in std_logic;
           puls : in std_logic;
           co_guardar: in std_logic;
           addrs : in integer range 0 to WIDTH*HEIGHT-1;
           co_addrs : in std_logic;
           co_suav : in std_logic;
```

```
           co_grad : in std_logic;
           co_ceros : in std_logic;
           co_ultima : in std_logic;
           co_adicional: in std_logic;
           co_1: in std_logic;
           fin_espera: in std_logic;
           imagen_ok: out std_logic;
           en_guardar: out std_logic;
           en_rom : out std_logic;
           en_ram : out std_logic;
           sel : out std_logic;
           guardarceros: out std_logic;
           en_addrs: out std_logic;
           en_suav : out std_logic;
           en_grad : out std_logic;
           en_ceros : out std_logic;
           en_ultima: out std_logic;
           --en_adicional: out std_logic;
           en_1: out std_logic;
           reset_addrs: out std_logic;
           reset_suav : out std_logic;
           reset_grad : out std_logic;
           reset_ceros : out std_logic;
           reset_ultima: out std_logic;
           reset_adicional: out std_logic;
           reset_rom: out std_logic;
           reset_ram: out std_logic;
           reset_buffer : out std_logic);
end control;

architecture Behavioral of control is
    type t_estados is (Reposo, Pulso, Espera, Guardar, NGyP, NGyS, CyS, GyS,
GyUL, Ultima, dchaS, izdaS, dchaG, izdaG, dchaSG, izdaSG, dchaSizdaG, izdaSdchaG,
Fin);
    signal estado_act, estado_sig: t_estados;

begin

    VarEstado: process (clk, reset)
    begin
        if reset = '1' then
            estado_act<=Reposo;
        elsif clk' event and clk='1' then
            estado_act<=estado_sig;
        end if;
    end process VarEstado;

    TransicionEstados: process (estado_act, puls, addrs, co_addrs, co_ceros,
co_suav, co_grad, co_adicional, co_ultima, co_guardar, co_1, fin_espera)
    begin
        estado_sig<=estado_act;
        case estado_act is
```

```
            when Reposo =>
                if puls='1' then
                    estado_sig<=Pulso;
                end if;

            when Pulso =>
                estado_sig<=Espera;

            when Espera =>
                if fin_espera='1' then
                    estado_sig<=Guardar;
                end if;

            when Guardar =>
                if co_guardar='1' then
                    estado_sig<=NGyP;
                end if;

            when NGyP =>
                if addrs=(2*WIDTH+5) then
                    estado_sig<=NGyS;
                end if;

            when NGyS =>
                if co_suav='1' then
                    estado_sig<=dchaS;
                elsif addrs=(4*WIDTH+13) then
                    estado_sig<=CyS;
                end if;

            when CyS =>
                if co_ceros='1' and co_suav='1' then
                    estado_sig<=dchaSizdaG;
                elsif co_ceros='1' then
                    estado_sig<=izdaG;
                elsif co_suav='1' then
                    estado_sig<=dchaS;
                end if;

            when GyS =>
                if co_grad='1' and co_suav='1' then
                    estado_sig<=dchaSG;
                elsif co_grad='1' then
                    estado_sig<=dchaG;
                elsif co_suav='1' then
                    estado_sig<=dchaS;
                end if;

            when GyUL =>
                if co_grad='1' then
                    estado_sig<=dchaG;
                end if;
```

```
            when Ultima =>
                if co_ceros='1' then
                    estado_sig<=Fin;
                end if;

            when dchaS =>
                if co_addrs='1' then
                    if co_grad='1' then
                        estado_sig<=dchaG;
                    else
                        estado_sig<=GyUL;
                    end if;
                else
                    if co_ceros='1' then
                        estado_sig<=izdaSG;
                    elsif co_grad='1' then
                        estado_sig<=izdaSdchaG;
                    else
                        estado_sig<=izdaS;
                    end if;
                end if;

            when izdaS=>
                if co_ceros='1' then
                    estado_sig<=izdaG;
                elsif co_grad='1' then
                    estado_sig<=dchaG;
                elsif addrs<(4*WIDTH+13) then
                    estado_sig<=NGyS;
                elsif addrs<(5*WIDTH+14) then
                    estado_sig<=CyS;
                else
                    estado_sig<=GyS;
                end if;

            when dchaG =>
                if co_addrs='1' then
                    if co_adicional='1' then
                        estado_sig<=Ultima;
                    elsif co_suav<='1' and co_1<='0' then
                        estado_sig<=dchaSizdaG;
                    else
                        estado_sig<=izdaG;
                    end if;
                else
                    if co_suav<='1' then
                        estado_sig<=dchaSizdaG;
                    else
                        estado_sig<=izdaG;
                    end if;
                end if;
```

```
        when izdaG =>
            if co_addrs='1' then
                if co_suav<='1' and co_1<='0' then
                    estado_sig<=dchaS;
                else
                    estado_sig<=GyUL;
                end if;
            else
                if co_suav<='1' then
                    estado_sig<=dchaSizdaG;
                else
                    estado_sig<=GyS;
                end if;
            end if;

        when izdaSG =>
            estado_sig<=GyS;

        when dchaSG =>
            if co_addrs='1' then
                estado_sig<=izdaG;
            else
                estado_sig<=izdaSG;
            end if;

        when dchaSizdaG =>
            if co_addrs='1' then
                estado_sig<=GyUL;
            else
                estado_sig<=izdaS;
            end if;

        when izdaSdchaG =>
            estado_sig<=izdaG;

        when Fin =>
            estado_sig<=Pulso;
    end case;
end process TransicionEstados;

Salidas: process (estado_act)
    begin
        case estado_act is
            when Reposo=>
                imagen_ok<='0';
                en_guardar<='0';
                en_rom<='0';
                en_ram <='0';
                sel<='0';
                guardarceros<='0';
                en_addrs<='0';
```

```
                    en_suav<='0';
                    en_grad <='0';
                    en_ceros <='0';
                    en_ultima<='0';
                    en_1<='0';
                    reset_addrs<='1';
                    reset_suav <='1';
                    reset_grad <='1';
                    reset_ceros <='1';
                    reset_ultima<='1';
                    reset_adicional<='1';
                    reset_rom<='1';
                    reset_ram<='1';
                    reset_buffer <='1';

                when Pulso =>
                    imagen_ok<='1';
                    en_guardar<='0';
                    en_rom<='0';
                    en_ram <='0';
                    sel<='0';
                    guardarceros<='0';
                    en_addrs<='0';
                    en_suav<='0';
                    en_grad <='0';
                    en_ceros <='0';
                    en_ultima<='0';
                    en_1<='0';
                    reset_addrs<='1';
                    reset_suav <='1';
                    reset_grad <='1';
                    reset_ceros <='1';
                    reset_ultima<='1';
                    reset_adicional<='1';
                    reset_rom<='1';
                    reset_ram<='1';
                    reset_buffer <='1';

                when Espera=>
                    imagen_ok<='0';
                    en_guardar<='0';
                    en_rom<='0';
                    en_ram <='0';
                    sel<='0';
                    guardarceros<='0';
                    en_addrs<='0';
                    en_suav<='0';
                    en_grad <='0';
                    en_ceros <='0';
                    en_ultima<='0';
                    en_1<='0';
                    reset_addrs<='1';
```

```
                    reset_suav <='1';
                    reset_grad <='1';
                    reset_ceros <='1';
                    reset_ultima<='1';
                    reset_adicional<='1';
                    reset_rom<='1';
                    reset_ram<='1';
                    reset_buffer <='1';

                when Guardar =>
                    imagen_ok<='0';
                    en_guardar<='1';
                    en_rom<='0';
                    en_ram <='0';
                    sel<='0';
                    guardarceros<='0';
                    en_addrs<='0';
                    en_suav<='0';
                    en_grad <='0';
                    en_ceros <='0';
                    en_ultima<='0';
                    en_1<='0';
                    reset_addrs<='1';
                    reset_suav <='1';
                    reset_grad <='1';
                    reset_ceros <='1';
                    reset_ultima<='1';
                    reset_adicional<='1';
                    reset_rom<='0';
                    reset_ram<='1';
                    reset_buffer <='1';

                when NGyP =>
                    imagen_ok<='0';
                    en_guardar<='0';
                    en_rom<='1';
                    en_ram <='0';
                    sel<='0';
                    guardarceros<='0';
                    en_addrs<='1';
                    en_suav<='0';
                    en_grad <='0';
                    en_ceros <='0';
                    en_ultima<='0';
                    en_1<='0';
                    reset_addrs<='0';
                    reset_suav <='1';
                    reset_grad <='1';
                    reset_ceros <='1';
                    reset_ultima<='1';
                    reset_adicional<='1';
                    reset_rom<='0';
```

```
                    reset_ram<='1';
                    reset_buffer <='0';

                when NGyS =>
                    imagen_ok<='0';
                    en_guardar<='0';
                    en_rom<='1';
                    en_ram <='0';
                    sel<='1';
                    guardarceros<='0';
                    en_addrs<='1';
                    en_suav<='1';
                    en_grad <='0';
                    en_ceros <='0';
                    en_ultima<='0';
                    reset_addrs<='0';
                    reset_suav <='0';
                    en_1<='0';
                    reset_grad <='1';
                    reset_ceros <='1';
                    reset_ultima<='1';
                    reset_adicional<='1';
                    reset_rom<='0';
                    reset_ram<='1';
                    reset_buffer <='0';

                when CyS =>
                    imagen_ok<='0';
                    en_guardar<='0';
                    en_rom<='1';
                    en_ram <='1';
                    sel<='1';
                    guardarceros<='0';
                    en_addrs<='1';
                    en_suav<='1';
                    en_grad <='0';
                    en_ceros <='1';
                    en_ultima<='0';
                    en_1<='0';
                    reset_addrs<='0';
                    reset_suav <='0';
                    reset_grad <='1';
                    reset_ceros <='0';
                    reset_ultima<='1';
                    reset_adicional<='1';
                    reset_rom<='0';
                    reset_ram<='0';
                    reset_buffer <='0';

                when GyS =>
                    imagen_ok<='0';
                    en_guardar<='0';
```

```
                en_rom<='1';
                en_ram <='1';
                sel<='1';
                guardarceros<='1';
                en_addrs<='1';
                en_suav<='1';
                en_grad <='1';
                en_ceros <='0';
                en_ultima<='0';
                en_1<='0';
                reset_addrs<='0';
                reset_suav <='0';
                reset_grad <='0';
                reset_ceros <='1';
                reset_ultima<='1';
                reset_adicional<='1';
                reset_rom<='0';
                reset_ram<='0';
                reset_buffer <='0';

            when GyUL =>
                imagen_ok<='0';
                en_guardar<='0';
                en_rom<='1';
                en_ram <='1';
                sel<='0';
                guardarceros<='1';
                en_addrs<='1';
                en_suav<='0';
                en_grad <='1';
                en_ceros <='0';
                en_ultima<='1';
                en_1<='0';
                reset_addrs<='0';
                reset_suav <='1';
                reset_grad <='0';
                reset_ceros <='1';
                reset_ultima<='0';
                reset_adicional<='0';
                reset_rom<='0';
                reset_ram<='0';
                reset_buffer <='0';

            when Ultima =>
                imagen_ok<='0';
                en_guardar<='0';
                en_rom<='1';
                en_ram <='1';
                sel<='0';
                guardarceros<='0';
                en_addrs<='1';
                en_suav<='0';
```

```
                    en_grad <='0';
                    en_ceros <='1';
                    en_ultima<='0';
                    en_1<='0';
                    reset_addrs<='0';
                    reset_suav <='1';
                    reset_grad <='1';
                    reset_ceros <='0';
                    reset_ultima<='1';
                    reset_adicional<='1';
                    reset_rom<='0';
                    reset_ram<='0';
                    reset_buffer <='0';

                when dchaS =>
                    imagen_ok<='0';
                    en_guardar<='0';
                    en_rom<='1';
                    en_ram <='1';
                    sel<='0';
                    guardarceros<='1';
                    en_addrs<='1';
                    en_suav<='0';
                    en_grad <='1';
                    en_ceros <='1';
                    en_ultima<='0';
                    en_1<='1';
                    reset_addrs<='0';
                    reset_suav <='1';
                    reset_grad <='0';
                    reset_ceros <='0';
                    reset_ultima<='1';
                    reset_adicional<='1';
                    reset_rom<='0';
                    reset_ram<='0';
                    reset_buffer <='0';

                when izdaS =>
                    imagen_ok<='0';
                    en_guardar<='0';
                    en_rom<='1';
                    en_ram <='1';
                    sel<='0';
                    guardarceros<='1';
                    en_addrs<='1';
                    en_suav<='0';
                    en_grad <='1';
                    en_ceros <='1';
                    en_ultima<='0';
                    en_1<='0';
                    reset_addrs<='0';
                    reset_suav <='1';
```

```
                reset_grad <='0';
                reset_ceros <='0';
                reset_ultima<='1';
                reset_adicional<='1';
                reset_rom<='0';
                reset_ram<='0';
                reset_buffer <='0';

            when dchaG =>
                imagen_ok<='0';
                en_guardar<='0';
                en_rom<='1';
                en_ram <='1';
                sel<='1';
                guardarceros<='0';
                en_addrs<='1';
                en_suav<='1';
                en_grad <='0';
                en_ceros <='0';
                en_ultima<='1';
                en_1<='0';
                reset_addrs<='0';
                reset_suav <='0';
                reset_grad <='1';
                reset_ceros <='1';
                reset_ultima<='0';
                reset_adicional<='0';
                reset_rom<='0';
                reset_ram<='0';
                reset_buffer <='0';

            when izdaG =>
                imagen_ok<='0';
                en_guardar<='0';
                en_rom<='1';
                en_ram <='1';
                sel<='1';
                guardarceros<='0';
                en_addrs<='1';
                en_suav<='1';
                en_grad <='0';
                en_ceros <='0';
                en_ultima<='1';
                en_1<='0';
                reset_addrs<='0';
                reset_suav <='0';
                reset_grad <='1';
                reset_ceros <='1';
                reset_ultima<='0';
                reset_adicional<='0';
                reset_rom<='0';
                reset_ram<='0';
```

```
                reset_buffer <='0';

            when dchaSG =>
                imagen_ok<='0';
                en_guardar<='0';
                en_rom<='1';
                en_ram <='1';
                sel<='0';
                guardarceros<='0';
                en_addrs<='1';
                en_suav<='0';
                en_grad <='0';
                en_ceros <='0';
                en_ultima<='0';
                en_1<='1';
                reset_addrs<='0';
                reset_suav <='1';
                reset_grad <='1';
                reset_ceros <='1';
                reset_ultima<='1';
                reset_adicional<='1';
                reset_rom<='0';
                reset_ram<='0';
                reset_buffer <='0';

            when izdaSG =>
                imagen_ok<='0';
                en_guardar<='0';
                en_rom<='1';
                en_ram <='1';
                sel<='0';
                guardarceros<='0';
                en_addrs<='1';
                en_suav<='0';
                en_grad <='0';
                en_ceros <='0';
                en_ultima<='0';
                en_1<='0';
                reset_addrs<='0';
                reset_suav <='1';
                reset_grad <='1';
                reset_ceros <='1';
                reset_ultima<='1';
                reset_adicional<='1';
                reset_rom<='0';
                reset_ram<='0';
                reset_buffer <='0';

            when izdaSdchaG =>
                imagen_ok<='0';
                en_guardar<='0';
                en_rom<='1';
```

```
                    en_ram <='1';
                    sel<='0';
                    guardarceros<='0';
                    en_addrs<='1';
                    en_suav<='0';
                    en_grad <='0';
                    en_ceros <='0';
                    en_ultima<='0';
                    en_1<='0';
                    reset_addrs<='0';
                    reset_suav <='1';
                    reset_grad <='1';
                    reset_ceros <='1';
                    reset_ultima<='1';
                    reset_adicional<='1';
                    reset_rom<='0';
                    reset_ram<='0';
                    reset_buffer <='0';

                when dchaSizdaG =>
                    imagen_ok<='0';
                    en_guardar<='0';
                    en_rom<='1';
                    en_ram <='1';
                    sel<='0';
                    guardarceros<='0';
                    en_addrs<='1';
                    en_suav<='0';
                    en_grad <='0';
                    en_ceros <='0';
                    en_ultima<='0';
                    en_1<='1';
                    reset_addrs<='0';
                    reset_suav <='1';
                    reset_grad <='1';
                    reset_ceros <='1';
                    reset_ultima<='1';
                    reset_adicional<='1';
                    reset_rom<='0';
                    reset_ram<='0';
                    reset_buffer <='0';

                when Fin =>
                    imagen_ok<='0';
                    en_guardar<='0';
                    en_rom<='0';
                    en_ram <='0';
                    sel<='0';
                    guardarceros<='0';
                    en_addrs<='0';
                    en_suav<='0';
                    en_grad <='0';
```

```
                    en_ceros <='0';
                    en_ultima<='0';
                    en_1<='0';
                    reset_addrs<='1';
                    reset_suav <='1';
                    reset_grad <='1';
                    reset_ceros <='1';
                    reset_ultima<='1';
                    reset_adicional<='1';
                    reset_rom<='1';
                    reset_ram<='1';
                    reset_buffer <='1';
            end case;
        end process Salidas;

end Behavioral;
```

Código 22. control.vhd en Canny

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity cont_addrs is
    generic(WIDTH: integer := 640;
            HEIGHT: integer := 480);

    Port ( clk : in std_logic;
           reset: in std_logic;
           enable : in std_logic;
           co : out std_logic;
           s: out integer range 0 to WIDTH*HEIGHT-1);
end cont_addrs;

architecture Behavioral of cont_addrs is
    signal contador: integer range 0 to WIDTH*HEIGHT-1;
begin
    process (clk, reset)
    begin
        if reset = '1' then
            contador<= 0;
        elsif clk' event and clk ='1' then
            if enable = '1' then
                if contador<(HEIGHT*WIDTH-1) then
                    contador<= contador+1;
                end if;
            end if;
        end if;
    end process;

    co<='1' when contador =(WIDTH*HEIGHT-1) else '0';
    s <= contador;

end Behavioral;
```

Código 23. cont_addrs.vhd en Canny

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity cont_suavizado is
    generic(WIDTH: integer := 640);

    Port ( clk : in STD_LOGIC;
           reset: in std_logic;
           enable : in STD_LOGIC;
           co : out STD_LOGIC);
end cont_suavizado;

architecture Behavioral of cont_suavizado is
    signal contador: integer range 0 to WIDTH-3;
begin
    process (clk, reset)
    begin
        if reset = '1' then
            contador<= 0;
        elsif clk' event and clk ='1' then
            if enable = '1' then
                if contador=(WIDTH-3) then
                    contador <= 0;
                else
                    contador<= contador+1;
                end if;
            end if;
        end if;
    end process;

    co<='1' when contador =(WIDTH-3) and enable='1' else '0';

end Behavioral;
```

Código 24. cont_suavizado.vhd en Canny

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity cont_ceros is
    generic(WIDTH: integer := 640);

    Port ( clk : in STD_LOGIC;
           reset: in std_logic;
           enable : in STD_LOGIC;
           co : out STD_LOGIC);
end cont_ceros;

architecture Behavioral of cont_ceros is
    signal contador: integer range 0 to WIDTH-1;
begin
    process (clk, reset)
    begin
        if reset = '1' then
            contador<= 0;
```

```
        elsif clk' event and clk ='1' then
            if enable = '1' then
                if contador=(WIDTH-1) then
                    contador <= 0;
                else
                    contador<= contador+1;
                end if;
            end if;
        end if;
    end process;

    co<='1' when contador =(WIDTH-1) and enable='1' else '0';


end Behavioral;
```

Código 25. cont_ceros.vhd en Canny

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity cont_grad is
    generic(WIDTH: integer := 640);

    Port ( clk : in STD_LOGIC;
           reset: in std_logic;
           enable : in STD_LOGIC;
           co : out STD_LOGIC);
end cont_grad;

architecture Behavioral of cont_grad is
    signal contador: integer range 0 to WIDTH-3;
begin
    process (clk, reset)
    begin
        if reset = '1' then
            contador<= 0;
        elsif clk' event and clk ='1' then
            if enable = '1' then
                if contador=(WIDTH-3) then
                    contador <= 0;
                else
                    contador<= contador+1;
                end if;
            end if;
        end if;
    end process;

    co<='1' when contador =(WIDTH-3) and enable='1' else '0';


end Behavioral;
```

Código 26. cont_grad.vhd en Canny

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity cont_ultima is
    generic(WIDTH: integer := 640);

    Port ( clk : in STD_LOGIC;
           reset: in std_logic;
           enable : in STD_LOGIC;
           co : out STD_LOGIC);
end cont_ultima;

architecture Behavioral of cont_ultima is
    signal contador: integer range 0 to WIDTH-1;
begin
    process (clk, reset)
    begin
        if reset = '1' then
            contador<= 0;
        elsif clk' event and clk ='1' then
            if enable = '1' then
                if contador=(WIDTH-1) then
                    contador <= 0;
                else
                    contador<= contador+1;
                end if;
            end if;
        end if;
    end process;

    co<='1' when contador =(WIDTH-1) and enable='1' else '0';


end Behavioral;
```

Código 27. cont_ultima.vhd en Canny

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity cont_1 is
    Port ( clk : in STD_LOGIC;
           enable : in STD_LOGIC;
           co : out STD_LOGIC);
end cont_1;

architecture Behavioral of cont_1 is
    signal contador: integer range 0 to 1;
begin
        process (clk)
    begin
        if clk'event and clk='1' then
            if enable = '1' then
                if contador<1 then
                    contador<= contador+1;
```

```
                end if;
            end if;
        end if;
    end process;

    co<='1' when contador =1 else '0';

end Behavioral;
```

Código 28. cont_1.vhd en Canny

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity cont_adicional is
    generic(WIDTH: integer := 640);

    Port ( clk : in STD_LOGIC;
           reset: in std_logic;
           enable : in STD_LOGIC;
           co : out STD_LOGIC);
end cont_adicional;

architecture Behavioral of cont_adicional is
    signal contador: integer range 0 to 4;          --es hasta 3 o hasta 4?
begin
    process (clk, reset)
    begin
        if reset = '1' then
            contador<= 0;
        elsif clk' event and clk ='1' then
            if enable = '1' then
                if contador<(4) then
                    contador<= contador+1;
                end if;
            end if;
        end if;
    end process;

    co<='1' when contador =(4) else '0';


end Behavioral;
```

Código 29. cont_adicional.vhd en Canny

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity canny is
    generic(WIDTH: integer := 640;
            HEIGHT: integer := 480;
            Talto : integer :=200;
            Tbajo: integer :=80);
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
```

```
           puls : in STD_LOGIC;
           pix_in : in STD_LOGIC_vector (7 downto 0);
           fin_espera: in std_logic;
           rd_addr: in  integer range 0 to WIDTH*HEIGHT-1;
           en_camara : out STD_LOGIC;
           rd_data : out std_logic);
end canny;

architecture Structural of canny is
   component rom
       generic(
            WIDTH : integer:=640;
            HEIGHT : integer:=480);
        Port ( clk : in std_logic;
               reset: in std_logic;
               pix_in: in std_logic_vector (7 downto 0);
               enable: in std_logic;
               en_guardar: in std_logic;
               addrs : in integer range 0 to WIDTH*HEIGHT-1;
               pix_out : out std_logic_vector(7 downto 0);
               co_guardar: out std_logic);
   end component;

   component line_buffer
        generic (
        WIDTH : integer := 640);
    port(
        clk: in std_logic;
        reset: in std_logic;
        pix_in: in std_logic_vector(7 downto 0);
        p00,p01,p02, p10, p11, p12, p20, p21, p22: out std_logic_vector(7 downto
0));
   end component;

   component suavizado
        generic(
            WIDTH : integer:=640;
            HEIGHT : integer:=480
        );
        Port ( clk : in std_logic;
               p00, p01, p02: in std_logic_vector (7 downto 0);
               p10, p11, p12: in std_logic_vector (7 downto 0);
               p20, p21, p22:in std_logic_vector (7 downto 0);
               pix_out : out std_logic_vector (7 downto 0));
   end component;

   component Mux
        Port ( a : in STD_LOGIC_vector (7 downto 0);
           b : in STD_LOGIC_vector (7 downto 0);
           sel : in STD_LOGIC;
           s : out STD_LOGIC_vector (7 downto 0));
   end component;
```

```
   component gradiente
        Port ( clk : in std_logic;
           p00, p01, p02: in std_logic_vector (7 downto 0);
           p10, p11, p12: in std_logic_vector (7 downto 0);
           p20, p21, p22: in std_logic_vector (7 downto 0);
           grad : out std_logic_vector (11 downto 0);
           angulo : out std_logic_vector (1 downto 0));
   end component;

   component line_buffer11
        generic (
        WIDTH : integer := 640);
        port(
            clk: in std_logic;
            reset: in std_logic;
            pix_in: in std_logic_vector(11 downto 0);
            p00,p01,p02, p10, p11, p12, p20, p21, p22: out std_logic_vector(11
downto 0));
   end component;

   component line_buffer1
        generic (
        WIDTH : integer := 640);
        port(
            clk: in std_logic;
            reset: in std_logic;
            pix_in: in std_logic_vector(1 downto 0);
            p00,p01,p02, p10, p11, p12, p20, p21, p22: out std_logic_vector(1
downto 0));
   end component;

   component supresion
        generic (Talto : integer :=1000;
                 Tbajo: integer :=255);
        Port ( clk : in std_logic;
               g00, g01, g02: in std_logic_vector (11 downto 0);
               g10, g11, g12: in std_logic_vector (11 downto 0);
               g20, g21, g22: in std_logic_vector (11 downto 0);
               a00, a01, a02: in std_logic_vector (1 downto 0);
               a10, a11, a12: in std_logic_vector (1 downto 0);
               a20, a21, a22: in std_logic_vector (1 downto 0);     --g de
gradiente y a de angulo
               tipoborde: out std_logic_vector (1 downto 0));
   end component;

   component histeresis
        Port ( clk : in std_logic;
           t00, t01, t02: in std_logic_vector (1 downto 0);
           t10, t11, t12: in std_logic_vector (1 downto 0);
           t20, t21, t22: in std_logic_vector (1 downto 0);
           borde : out std_logic);
```

```
end component;

component puerta_and
     Port ( a : in STD_LOGIC;
        b : in STD_LOGIC;
        s : out STD_LOGIC);
end component;

component ram
     generic(
         WIDTH : integer:=640;
         HEIGHT : integer:=480);
     Port ( clk : in std_logic;
            enable : in std_logic;
            reset : in std_logic;
            pix_in: in std_logic;
            rd_addr : in  integer range 0 to WIDTH*HEIGHT-1;
            rd_data : out std_logic);
end component;

component control
     generic(WIDTH: integer := 640;
         HEIGHT: integer := 480);
     Port ( clk : in std_logic;
            reset : in std_logic;
            puls : in std_logic;
            co_guardar: in std_logic;
            addrs : in integer range 0 to WIDTH*HEIGHT-1;
            co_addrs : in std_logic;
            co_suav : in std_logic;
            co_grad : in std_logic;
            co_ceros : in std_logic;
            co_ultima : in std_logic;
            co_adicional: in std_logic;
            co_1: in std_logic;
            fin_espera: in std_logic;
            imagen_ok: out std_logic;
            en_guardar: out std_logic;
            en_rom : out std_logic;
            en_ram : out std_logic;
            sel : out std_logic;
            guardarceros: out std_logic;
            en_addrs: out std_logic;
            en_suav : out std_logic;
            en_grad : out std_logic;
            en_ceros : out std_logic;
            en_ultima: out std_logic;
            en_1: out std_logic;
            reset_addrs: out std_logic;
            reset_suav : out std_logic;
            reset_grad : out std_logic;
            reset_ceros : out std_logic;
```

```
            reset_ultima: out std_logic;
            reset_adicional: out std_logic;
            reset_rom: out std_logic;
            reset_ram: out std_logic;
            reset_buffer : out std_logic);
end component;

component cont_addrs
     generic(WIDTH: integer := 640;
         HEIGHT: integer := 480);
     Port ( clk : in std_logic;
        reset: in std_logic;
        enable : in std_logic;
        co : out std_logic;
        s: out integer range 0 to WIDTH*HEIGHT-1);
end component;

component cont_ceros
     generic(WIDTH: integer := 640);
     Port ( clk : in STD_LOGIC;
        reset: in std_logic;
        enable : in STD_LOGIC;
        co : out STD_LOGIC);
end component;

component cont_grad
     generic(WIDTH: integer := 640);
     Port ( clk : in STD_LOGIC;
        reset: in std_logic;
        enable : in STD_LOGIC;
        co : out STD_LOGIC);
end component;

component cont_suavizado
     generic(WIDTH: integer := 640);
     Port ( clk : in STD_LOGIC;
        reset: in std_logic;
        enable : in STD_LOGIC;
        co : out STD_LOGIC);
end component;

component cont_ultima
     generic(WIDTH: integer := 640);

     Port ( clk : in STD_LOGIC;
            reset: in std_logic;
            enable : in STD_LOGIC;
            co : out STD_LOGIC);
end component;

component cont_adicional
     generic(WIDTH: integer := 640);
```

```
        Port ( clk : in STD_LOGIC;
               reset: in std_logic;
               enable : in STD_LOGIC;
               co : out STD_LOGIC);
   end component;

   component cont_1
        Port ( clk : in STD_LOGIC;
           enable : in STD_LOGIC;
           co : out STD_LOGIC);
   end component;

    signal out_rom: std_logic_vector(7 downto 0);
    signal p00s, p01s, p02s, p10s, p11s, p12s, p20s, p21s, p22s: std_logic_vector
(7 downto 0);
    signal out_suav: std_logic_vector (7 downto 0);
    signal sel: std_logic;
    signal in_grad: std_logic_vector (7 downto 0);
    signal p00g, p01g, p02g, p10g, p11g, p12g, p20g, p21g, p22g: std_logic_vector
(7 downto 0);
    signal out_g_g: std_logic_vector (11 downto 0);
    signal out_g_a: std_logic_vector (1 downto 0);
    signal g00, g01, g02, g10, g11, g12, g20, g21, g22: std_logic_vector (11
downto 0);
    signal a00, a01, a02, a10, a11, a12, a20, a21, a22: std_logic_vector (1
downto 0);
    signal out_supresion: std_logic_vector (1 downto 0);
    signal t00, t01, t02, t10, t11, t12, t20, t21, t22: std_logic_vector (1
downto 0);
    signal out_his, in_ram: std_logic;
    signal guardarceros: std_logic;
    signal addrs: integer range 0 to WIDTH*HEIGHT-1;
    signal en_guardar, en_rom, en_ram, en_addrs, en_ceros, en_grad, en_suavizado,
en_ultima, en_adicional, en_1a, en_1: std_logic;
    signal reset_rom, reset_ram, reset_linebuffer, reset_addrs, reset_ceros,
reset_grad, reset_suavizado, reset_ultima, reset_adicional: std_logic;
    signal co_guardar, co_addrs, co_ceros, co_grad, co_suavizado, co_ultima,
co_adicional, co_1: std_logic;

begin

    i1: rom
        generic map(
            WIDTH=>WIDTH,
            HEIGHT=>HEIGHT)
        port map(
            clk=>clk,
            reset=>reset_rom,
            pix_in=>pix_in,
            enable=>en_rom,
            en_guardar=>en_guardar,
```

```
            addrs=>addrs,
            pix_out=>out_rom,
            co_guardar=>co_guardar);

    i2: line_buffer
        generic map(
            WIDTH=>WIDTH)
        port map(
            clk=>clk,
            reset=>reset_linebuffer,
            pix_in=>out_rom,
            p00=>p00s,
            p01=>p01s,
            p02=>p02s,
            p10=>p10s,
            p11=>p11s,
            p12=>p12s,
            p20=>p20s,
            p21=>p21s,
            p22=>p22s);

    i3: suavizado
        generic map(
            WIDTH=>WIDTH,
            HEIGHT=>HEIGHT)
        port map(
            clk=>clk,
            p00=>p00s,
            p01=>p01s,
            p02=>p02s,
            p10=>p10s,
            p11=>p11s,
            p12=>p12s,
            p20=>p20s,
            p21=>p21s,
            p22=>p22s,
            pix_out=>out_suav
        );

    i4: Mux
        port map(
            a=>out_suav,
            b=>p11s,
            sel=>sel,
            s=>in_grad);

    i5: line_buffer
        generic map(
            WIDTH=>WIDTH)
        port map(
            clk=>clk,
            reset=>reset_linebuffer,
```

```
            pix_in=>in_grad,
            p00=>p00g,
            p01=>p01g,
            p02=>p02g,
            p10=>p10g,
            p11=>p11g,
            p12=>p12g,
            p20=>p20g,
            p21=>p21g,
            p22=>p22g);

    i6: gradiente
        port map(
            clk=>clk,
            p00=>p00g,
            p01=>p01g,
            p02=>p02g,
            p10=>p10g,
            p11=>p11g,
            p12=>p12g,
            p20=>p20g,
            p21=>p21g,
            p22=>p22g,
            grad=>out_g_g,
            angulo=>out_g_a);

    i7: line_buffer11
        generic map(
            WIDTH=>WIDTH)
        port map(
            clk=>clk,
            reset=>reset_linebuffer,
            pix_in=>out_g_g,
            p00=>g00,
            p01=>g01,
            p02=>g02,
            p10=>g10,
            p11=>g11,
            p12=>g12,
            p20=>g20,
            p21=>g21,
            p22=>g22);

    i8: line_buffer1
        generic map(
            WIDTH=>WIDTH)
        port map(
            clk=>clk,
            reset=>reset_linebuffer,
            pix_in=>out_g_a,
            p00=>a00,
            p01=>a01,
```

```
            p02=>a02,
            p10=>a10,
            p11=>a11,
            p12=>a12,
            p20=>a20,
            p21=>a21,
            p22=>a22);

    i9: supresion
        generic map(
            Talto =>Talto,
            Tbajo=>Tbajo)
        port map(
            clk=>clk,
            g00=>g00,
            g01=>g01,
            g02=>g02,
            g10=>g10,
            g11=>g11,
            g12=>g12,
            g20=>g20,
            g21=>g21,
            g22=>g22,
            a00=>a00,
            a01=>a01,
            a02=>a02,
            a10=>a10,
            a11=>a11,
            a12=>a12,
            a20=>a20,
            a21=>a21,
            a22=>a22,
            tipoborde=>out_supresion);

    i10: line_buffer1
        generic map(
            WIDTH=>WIDTH)
        port map(
            clk=>clk,
            reset=>reset_linebuffer,
            pix_in=>out_supresion,
            p00=>t00,
            p01=>t01,
            p02=>t02,
            p10=>t10,
            p11=>t11,
            p12=>t12,
            p20=>t20,
            p21=>t21,
            p22=>t22);

    i11: histeresis
```

```
        port map(
            clk=>clk,
            t00=>t00,
            t01=>t01,
            t02=>t02,
            t10=>t10,
            t11=>t11,
            t12=>t12,
            t20=>t20,
            t21=>t21,
            t22=>t22,
            borde=>out_his);

    i12: puerta_and
        port map(
            a=>out_his,
            b=>guardarceros,
            s=>in_ram);

    i13: ram
        generic map(
            WIDTH=>WIDTH,
            HEIGHT=>HEIGHT)
        port map(
            clk=>clk,
            enable=>en_ram,
            reset=>reset_ram,
            pix_in=>in_ram,
            rd_addr=>rd_addr,
            rd_data=>rd_data);

    i14: control
        generic map(
            WIDTH=>WIDTH,
            HEIGHT=>HEIGHT)
        port map(
            clk=>clk,
            reset=>reset,
            puls=>puls,
            co_guardar=>co_guardar,
            addrs=>addrs,
            co_addrs=>co_addrs,
            co_suav=>co_suavizado,
            co_grad=>co_grad,
            co_ceros=>co_ceros,
            co_ultima=>co_ultima,
            co_adicional=>co_adicional,
            co_1=>co_1,
            fin_espera=> fin_espera,
            imagen_ok=> en_camara,
            en_guardar=>en_guardar,
            en_rom=>en_rom,
```

```
            en_ram=>en_ram,
            sel=>sel,
            guardarceros=>guardarceros,
            en_addrs=>en_addrs,
            en_suav=>en_suavizado,
            en_grad=>en_grad,
            en_ceros=>en_ceros,
            en_ultima=>en_ultima,
            en_1=>en_1a,
            reset_addrs=>reset_addrs,
            reset_suav=>reset_suavizado,
            reset_grad=>reset_grad,
            reset_ceros=>reset_ceros,
            reset_ultima=>reset_ultima,
            reset_adicional=>reset_adicional,
            reset_rom=>reset_rom,
            reset_ram=>reset_ram,
            reset_buffer=>reset_linebuffer);

    i15: cont_addrs
        generic map(
            WIDTH=>WIDTH,
            HEIGHT=>HEIGHT)
        port map(
            clk=>clk,
            reset=>reset_addrs,
            enable=>en_addrs,
            co=>co_addrs,
            s=>addrs);

    i16: cont_suavizado
        generic map(
            WIDTH=>WIDTH)
        port map(
            clk=>clk,
            reset=>reset_suavizado,
            enable=>en_suavizado,
            co=>co_suavizado);

    i17: cont_ceros
        generic map(
            WIDTH=>WIDTH)
        port map(
            clk=>clk,
            reset=>reset_ceros,
            enable=>en_ceros,
            co=>co_ceros);

    i18: cont_grad
        generic map(
            WIDTH=>WIDTH)
        port map(
```

```
            clk=>clk,
            reset=>reset_grad,
            enable=>en_grad,
            co=>co_grad);

    i19: cont_ultima
        generic map(
            WIDTH=>WIDTH)
        port map(
            clk=>clk,
            reset=>reset_ultima,
            enable=>en_ultima,
            co=>co_ultima);

    i20: puerta_and
        port map(
            a=>en_1a,
            b=>co_addrs,
            s=>en_1);

    i21: cont_1
        port map(
            clk=>clk,
            enable=>en_1,
            co=>co_1);

    i22: puerta_and
        port map(
            a=>co_1,
            b=>co_grad,
            s=>en_adicional);

    i23: cont_adicional
        generic map(
            WIDTH=>WIDTH)
        port map(
            clk=>clk,
            reset=>reset_adicional,
            enable=>en_adicional,
            co=>co_adicional);


end Structural;
```

Código 30. canny.vhd en Canny

- **TESTBENCHES**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity tb_sobel is
end tb_sobel;

architecture Behavioral of tb_sobel is
    constant WIDTH  : integer := 640;
    constant HEIGHT : integer := 480;


    signal clk_tb      : std_logic := '0';
    signal reset_tb    : std_logic := '1';
    signal puls_tb     : std_logic := '0';
    signal pix_in_tb   : std_logic_vector (7 downto 0):= (others => '0');
    signal fin_espera_tb: std_logic :='0';
    signal rd_addr_tb: integer range 0 to WIDTH*HEIGHT-1:=0;
    signal en_camara_tb: std_logic;
    signal rd_data_tb: std_logic;

    constant clk_period : time := 10 ns;
begin

    DUT: entity work.sobel(structural)
        generic map(
            WIDTH  => WIDTH,
            HEIGHT => HEIGHT
        )
        port map(
            clk     => clk_tb,
            reset => reset_tb,
            puls    => puls_tb,
            pix_in  => pix_in_tb,
            fin_espera=> fin_espera_tb,
            rd_addr => rd_addr_tb,
            en_camara=>en_camara_tb,
            rd_data=>rd_data_tb
        );

    clk_process : process
    begin
        clk_tb <= '0';
        wait for clk_period/2;
        clk_tb <= '1';
        wait for clk_period/2;
    end process;

    stim_proc: process

        variable fila    : integer;
```

```
        variable columna : integer;

    begin
        reset_tb <= '1';
        wait for 20 ns;

        reset_tb <= '0';
        wait for 20 ns;

        puls_tb <='0';
        wait for 20 ns;

        puls_tb<='1';
        wait for clk_period;

        puls_tb<='0';

        wait until rising_edge(clk_tb) and en_camara_tb='1';
        wait for 45 ns;                                          --tiempo que
tarda la camara en capturar la imagen
        fin_espera_tb<='1';
        wait for clk_period;
        fin_espera_tb<='0';

        for fila in 0 to HEIGHT-1 loop
            for columna in 0 to WIDTH-1 loop
                -------------------------------------------------------
                -- Columna negra
                -------------------------------------------------------
                if columna = 3 then
                    pix_in_tb <= x"00";

                -------------------------------------------------------
                -- Resto blanco
                -------------------------------------------------------
                else
                    pix_in_tb <= x"FF";

                end if;

                wait until rising_edge(clk_tb);
            end loop;
        end loop;
        wait;
    end process;
end Behavioral;
```

Código 31. Testbench para el proyecto Sobel (tb_sobel.vhd)

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;

entity tb_canny is
end tb_canny;

architecture Behavioral of tb_canny is
    constant WIDTH  : integer := 640;
    constant HEIGHT : integer := 480;

    signal clk_tb      : std_logic := '0';
    signal reset_tb    : std_logic := '1';
    signal puls_tb     : std_logic := '0';
    signal pix_in_tb   : std_logic_vector (7 downto 0):= (others => '0');
    signal fin_espera_tb: std_logic :='0';
    signal rd_addr_tb: integer range 0 to WIDTH*HEIGHT-1:=0;
    signal en_camara_tb: std_logic;
    signal rd_data_tb: std_logic;

    constant clk_period : time := 10 ns;
begin


    DUT: entity work.canny(structural)
        generic map(
            WIDTH  => WIDTH,
            HEIGHT => HEIGHT
        )
        port map(
            clk     => clk_tb,
            reset => reset_tb,
            puls    => puls_tb,
            pix_in  => pix_in_tb,
            fin_espera=> fin_espera_tb,
            rd_addr=>rd_addr_tb,
            en_camara=>en_camara_tb,
            rd_data=>rd_data_tb
        );

    clk_process : process
    begin
        clk_tb <= '0';
        wait for clk_period/2;
        clk_tb <= '1';
        wait for clk_period/2;
    end process;

    stim_proc: process

        variable fila    : integer;
        variable columna : integer;

    begin
```

```
        reset_tb <= '1';
        wait for 20 ns;

        reset_tb <= '0';
        wait for 20 ns;

        puls_tb <='0';
        wait for 20 ns;

        puls_tb<='1';
        wait for clk_period;

        puls_tb<='0';

        wait until rising_edge(clk_tb) and en_camara_tb='1';  --1

        --wait until rising_edge(clk_tb);    --2
        --wait until rising_edge(clk_tb);    --3
        wait for 45 ns;
        fin_espera_tb<='1';
        wait for clk_period;
        fin_espera_tb<='0';

        --wait until rising_edge(clk_tb);

        for fila in 0 to HEIGHT-1 loop
            for columna in 0 to WIDTH-1 loop
                ------------------------------------------------------
                -- Columna negra
                ------------------------------------------------------
                if columna = 3 then
                    pix_in_tb <= x"00";

                ------------------------------------------------------
                -- Resto blanco
                ------------------------------------------------------
                else
                    pix_in_tb <= x"FF";

                end if;

                wait until rising_edge(clk_tb);
            end loop;
        end loop;
        wait;
    end process;
end Behavioral;
```

Código 32. Testbench para el proyecto Canny (tb_canny.vhd)