Universidad de Sevilla.
Escuela Politécnica Superior de Sevilla.

Trabajo Fin de Grado en

Ingeniería Electrónica Industrial.

Procesado de una señal de vídeo para la detección de bordes en tiempo real sobre FPGAs.

Autores: Alejandro Aguilocho Domínguez

Tutores: Carlos Jesús Jiménez Fernández

Pilar Parra Fernández

Departamento de Tecnología Electrónica

Junio 2018

Universidad de Sevilla.
Escuela Politécnica Superior de Sevilla.

RESUMEN

Procesado de una señal de vídeo para la detección de bordes en tiempo real sobre FPGAs.

Autor: Alejandro Aguilocho Domínguez

Tutores: Carlos Jesús Jiménez Fernández

Pilar Parra Fernández

En este Trabajo Fin de Grado se ha implementado un sistema basado en FPGA el cual realiza el procesado para la detección de bordes de una señal de video obtenida a través de una cámara y mostrando el resultado en cualquier pantalla que disponga de tecnología HDMI o transmitiendo instantáneas a un ordenador.

Para la realización de este proyecto se ha requerido del análisis de un diseño base proporcionado por el fabricante de la placa de desarrollo para la comprensión de su funcionamiento, ya que ha sido sobre el cual se ha implementado el sistema objetivo.

Desde el punto de vista teórico ha sido necesario estudiar y comprender la estructura de una imagen y el procesado de estas, además de, interfaces avanzadas como el uso de memorias DDR, control y adquisición de datos de una cámara y generación de señales HDMI.

Universidad de Sevilla.
Escuela Politécnica Superior de Sevilla.

ABSTRACT

Real-time video signal processing on FPGA for edge detection

Author: Alejandro Aguilocho Domínguez

Tutors: Carlos Jesús Jiménez Fernández

Pilar Parra Fernández

In this End-Of-Degree project a FPGA based system that processes a video signal and performs real-time edge detection has been implemented. The video output is displayed on a screen via an HDMI interface and it is also possible to transmit image captures to a computer.

In order to carry out this project, first of all, the analysis of the base design supplied by the manufacturer of the board was developed. This design only shows the video coming from a video camera on an HDMI screen. A second task has been to modify this base design to introduce the ability to process the video signal and perform edge detection.

Among the learning objects of this work are, in addition to the VHDL FPGAs design, the study of the structure of an image, and the way in which the image is transmitted from the video camera to the FPGA, the signal processing algorithms for edge detection and the operating mode of the interface of a DDR memory with the HDMI protocol.

Universidad de Sevilla.
Escuela Politécnica Superior de Sevilla.

Trabajo Fin de Grado en

Ingeniería Electrónica Industrial.

Procesado de una señal de vídeo para la detección de bordes en tiempo real sobre FPGAs.

Memoria Descriptiva

Autor: Alejandro Aguilocho Domínguez

Tutores: Carlos Jesús Jiménez Fernández

Pilar Parra Fernández

Departamento de Tecnología Electrónica

Junio 2018

Índice

Índice de Figuras

Índice de Tablas

1. INTRODUCCIÓN

En la actualidad está cobrando cada vez más importancia el campo de la Visión Artificial. Esta se define como un campo de la Inteligencia Artificial que, mediante la utilización de las técnicas adecuadas, permite la obtención, procesamiento y análisis de cualquier tipo de información espacial obtenida a

Un aspecto interesante dentro del campo de la visión artificial es la detección de bordes. Esta tiene numerosas aplicaciones en distintos campos. Por ejemplo, en medicina, es utilizada en el procesado de imágenes clínicas para la delimitación precisa de tumores. En sistemas biométricos es utilizada para la detección de los contornos de iris del ojo, reconocimiento facial, etc. En vigilancia, facilita la detección de objetos o personas en movimiento. En la industria, su uso más tradicional, la detección de bordes es usada en metrología para determinar profundidad, forma, tamaño y otras propiedades de las piezas, facilitando el control de calidad.

Cada día su uso es cada vez más intensivo y requiere de más resolución e imágenes por segundo, lo que se traduce en una mayor tasa de datos.

Aunque hoy día la mayoría de los sistemas de visión artificial hacen uso de procesadores de propósito general, el procesamiento de imágenes a alta velocidad impone muchas limitaciones con el uso único de procesadores.

En este momento es cuando entran en escena los dispositivos lógicos programables, como las FPGA. Dado su carácter paralelo, son capaces de realizar múltiples tareas al mismo tiempo, por lo que resultan idóneas para procesos repetitivos a alta velocidad.

No debemos de confundir el papel de una FPGA en el procesado de imágenes en tiempo real, esta no será la encargada de detectar por ejemplo un objeto en una imagen. Una FPGA trabajaría como apoyo a un procesador encargándose del preprocesado de una imagen, disminuyendo la carga de trabajo del procesador y aumentando la tasa de datos capaz de procesar el sistema en tiempo real.

En este trabajo se da un pequeño paso introductorio a esta tecnología. El principal objetivo de este trabajo es la adquisición de imágenes en tiempo real de una cámara conectada a una placa de desarrollo que integra una FPGA, almacenarla en memoria DDR, procesarla mediante un filtro de detección de bordes y transmitirla vía HDMI para poder ser visualizada en una pantalla.

Para abordar adecuadamente estos objetivos antes han de comprenderse multitud de conceptos, los cuales han sido objetivos complementarios necesarios, pero no menos importantes:

- **Teoría de imágenes.**
  Comprender que es una imagen, cómo se caracteriza, formatos de color, etc.
- **Filtros de imagen.**
  Tipos de filtros de imágenes e implementación.
- **FPGA.**
  Entender cómo funciona y de que se componen, llegando a bajo nivel, para así determinar sus capacidades y limitaciones.
- **Memorias de alta velocidad. DDR.**
  Modo de funcionamiento y métodos de control de este tipo de memorias tan extendidas y rápidas.
- **Señales de video digital.**
  Protocolos de envío de imágenes digitales a alta velocidad.

El trabajo está dividido en tres grandes capítulos. En el primero, “Sistemas de procesado de imagen” se introducen los aspectos más teóricos del trabajo, estos son: señales de video, imágenes, filtros y FPGA, junto con el lenguaje de descripción utilizado (VHDL) y el entorno de desarrollo Xilinx ISE.

En el segundo capítulo, “Sistema de detección de bordes. Características y componentes” son explicados los aspectos técnicos del trabajo: placa de desarrollo y cámara utilizada, diseño de ejemplo del que ha partido el trabajo y cómo se realiza la interfaz con la cámara, memoria y señal de video.

2. SISTEMAS DE PROCESADO DE IMAGEN

En esta sección nos centraremos en los aspectos más importantes de los sistemas de procesado de imagen, ya sea teoría sobre las imágenes y cómo estas se definen, cómo los aspectos matemáticos de los procesos de filtrado de imagen.

También se describirá brevemente lo que es una FPGA y cómo funciona internamente.

Por último, se explica el lenguaje de diseño utilizado, VHDL, y el entorno de desarrollo necesario para la configuración de la FPGA, Xilinx ISE, así como su metodología de diseño.

2.1. Señal de vídeo

El video es un sistema o tecnología que por medios electrónicos permite grabar, copiar, reproducir y retransmitir secuencias de imágenes en movimiento que pueden estar acompañadas con sonido (contenido multimedia).

Los sistemas de video se caracterizan por su resolución, relación de aspecto, refresco, capacidades de color y otras cualidades. Actualmente existen dos tipos de video: analógico y digital. Aunque el video analógico es más antiguo, el video digital, por sus múltiples ventajas, es hoy día mucho más usado que el analógico.

- **Video analógico**

  Una señal de video analógico es aquella que es transferida por una señal analógica. Sus componentes son la luminancia (Y), el brillo y la crominancia (C).

  Cuando estas componentes están combinadas en un canal, se utiliza el término *video compuesto*. Este es el caso, entre otros, de los formatos estándares NTSC, PAL y SECAM.

  El video analógico también puede transportarse en canales separados, esto proporciona mayor calidad. Es el caso de S-Video (Separated-Video), ver figura b, que utiliza dos canales separados para luminancia y crominancia para minimizar las interferencias, y en formatos de video de componentes multicanal.

Video compuesto: único canal, por conector RCA
(a)

S-Video
(b)

Video por componentes:
las señales rojo, verde y azul se tratan por separado
(c)

Euroconector: conector multicanal que permite canales YC, video compuesto y RGB
(d)

VGA: transporta video RGB e información de sincronización vertical y horizontal
(f)

D-Terminal: usado en dispositivos japoneses
(g)

- **Vídeo digital**

Un video digital es aquel que representa las imágenes en movimiento mediante datos digitales codificados. Los formatos de video de carácter digital proporcionan mayor calidad en la señal de video, así como, menores interferencias y ruido, mayor resolución y posibilidades añadidas como metadatos que proporcionan información extra. Asimismo, puede ser copiado sin pérdida de calidad.

Todas las interfaces se caracterizan por una transmisión digital serie síncrona multicanal a grandes frecuencias.

Algunas de las interfaces estándares mas comunes son: Digital Video Interface (DVI), High Definition Multimedia Interface (HDMI) y Display Port (DP).

2.2. Introducción al procesado de imágenes

2.2.1. Imagen digital

Una imagen puede ser definida como una función bidimensional, $f(x,y)$ donde x e y son coordenadas espaciales, y f, la amplitud de cualquier par de coordenadas, la intensidad o nivel de gris en ese punto. Cuando los valores de las coordenadas y la amplitud son discretos podemos denominar la imagen digital (Gonzalez & Woods, 2007).

Cada par de coordenadas con su amplitud asociada se llama *elemento de imagen*, *pel* o más comúnmente, *pixel*.

Una imagen digital $f(x,y)$ se representa como una matriz de M filas y N columnas de valores discretos. Cómo se muestra en la figura 1, por convención tomamos comol origen la coordenada superior izquierda.

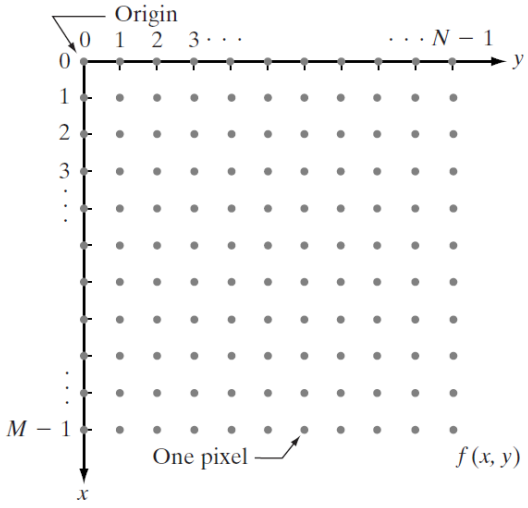

Figura 1. Convención de coordenadas de una imagen.

Gracias a la notación introducida, podemos escribir una imagen digital $M\ x\ N$ de la siguiente forma de matriz compacta:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & \dots & f(1,N-1) \\ \dots & \dots & \dots & \dots \\ f(M-1,0) & f(M-1,1) & \dots & f(M-1,N-1) \end{bmatrix}$$

2.2.1.1. Relaciones de vecindad.

Una imagen la denotaremos como $f(x,y)$, los puntos los denotaremos por letras minúsculas p o q y denotaremos por S un conjunto de puntos de la imagen $f(x,y)$.

- Vecinos de un pixel

Un punto p de coordenadas (x,y) tiene 4 vecinos verticales y horizontales cuyas coordenadas son:

$$(x+1,y),(x-1,y),(x,y+1),(x,y-1)$$

A este conjunto de puntos se le llama los 4-vecinos de p y lo denotaremos como $N_4(p)$.

Los 4 vecinos diagonales de p tienen como coordenadas:

$$(x+1,y+1),(x+1,y-1),(x-1,y+1),(x-1,y-1)$$

y los denotaremos como $N_D(p)$. A estos 4 vecinos diagonales junto con $N_4(p)$ le llamaremos los 8-vecinos de p, denotados como $N_8(p)$. Sus coordenadas son:

$(x-1,y-1)$	$(x,y-1)$	$(x+1,y-1)$
$(x-1,y)$	(x,y)	$(x+1,y)$
$(x-1,y+1)$	$(x,y+1)$	$(x+1,y+1)$

2.2.2. Espacio de color

Un modelo de color es un modelo matemático abstracto en el que los colores pueden representarse como listas de números, normalmente como tres o cuatro valores o componentes de color (ej. RGB y CMYK son modelos de color).

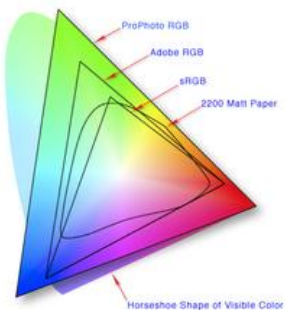

Figura 2. Cromaticidades en un espacio de color.

Añadiendo una función de mapeo entre el modelo de color y un espacio de color de referencia se obtiene una "huella" en el espacio de color de referencia. A esta "huella" se la conoce como gama de color y, en combinación con el modelo de color, define un nuevo espacio de color. Por ejemplo, Adobe RGB y sRGB son dos espacios de color absolutos diferentes basados en el modelo RGB. En la figura 2 podemos ver varios ejemplos de distintos espacios de color.

Un modelo de color que no tiene asociada una función de mapeo a un espacio de color absoluto es más o menos un sistema de color arbitrario sin conexión a un sistema de interpretación de color.

En el sentido más genérico de la definición dada, los espacios de color se pueden definir sin el uso de un modelo de color. Estos espacios, como Pantone, son un conjunto de nombres o números definidos por la existencia de un conjunto correspondiente de muestras de color físico.

- **RGB**

  RGB es un modelo de color basado en la síntesis aditiva, con el que es posible representar un color mediante la mezcla por adición de los tres colores de luz primarios. El modelo de color RGB no define por sí mismo lo que significa exactamente rojo, verde o azul, por lo que los mismos valores RGB pueden mostrar colores notablemente diferentes en distintos dispositivos que usen este modelo de color. Aunque utilicen un mismo modelo de color, sus espacios de color pueden variar considerablemente.

- **YUV**

  YUV es un espacio de color típicamente usado como parte de un sistema de procesamiento de imagen en color. Una imagen o vídeo en color se codifica en este espacio de color teniendo en cuenta la percepción humana, lo que permite un ancho de banda reducido para los componentes de diferencia de color o crominancia, de esta forma, hace que los errores de transmisión o las imperfecciones de compresión se oculten más eficientemente a la percepción humana que usando una representación RGB directa. Otros espacios de color tienen propiedades similares y la principal razón para implementar o investigar las propiedades de YUV o de su similar, Y'UV se encuentran tanto las de interactuar con televisión analógica o digital o con equipo fotográfico que sea conforme a ciertos estándares de este espacio.

El ámbito de los términos YUV, Y'UV, YCbCr y YPbPr es a veces ambiguo y coincidente. Históricamente, los espacios YUV y Y'UV fueron usados para la codificación analógica específica del color en sistemas de televisión, mientras que YCbCr fue usado para la codificación digital adecuada de la información de color para la compresión y transmisión de video e imagen fija, tal como ocurre con las normas MPEG y JPEG, respectivamente. Hoy en día, el término es comúnmente usado en la industria de la computación para describir los formatos de archivo que son codificados usando el espacio de color YCbCr.

En la figura 3 podemos apreciar un plano de color U-V. Se representa como un mapa de coordenadas, en el que con dos coordenadas, U y V, determinamos cualquier color de la escala cromática.

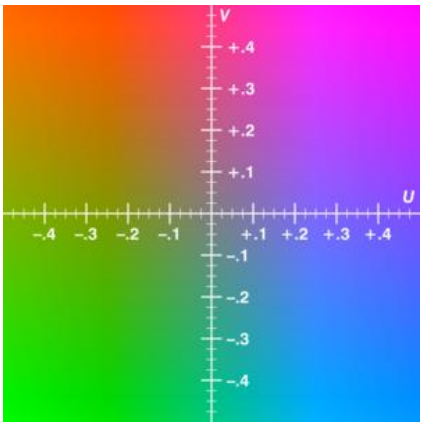

Figura 3. Ejemplo de plano de color U-V.

El modelo Y'UV, usado en los sistemas de vídeo compuesto de color PAL y NTSC, define un espacio de color en términos de una componente de luminancia y dos componentes de crominancia (UV). Los sistemas monocromáticos anteriores usaban solamente la información de luma. La información de color fue añadida por separado mediante la modulación de una subportadora, de modo que un receptor de blanco y negro pudiera ser capaz de recibir una transmisión de color en el formato monocromático propio del equipo. El símbolo Y' denota la señal de luminancia con corrección gamma.

El espacio de color YPbPr usado en video componente analógico y su versión digital YCbCr se derivan de Y'UV y a veces son llamados así. El espacio de color YIQ usado en el estándar de televisión cromática estadounidense NTSC también está relacionado con Y'UV aunque de manera más compleja.

2.2.3. Procesado digital

El procesado o tratado digital de una imagen es el conjunto de técnicas que se aplican con el objetivo de mejorar la calidad o facilitar la búsqueda de información.

Los principales objetivos de la aplicación de filtros son:

- Suavizar la imagen: reducir la cantidad de variaciones de intensidad entre píxeles vecinos.
- Eliminar ruido: descartar aquellos píxeles cuyo nivel de intensidad es muy diferente al de sus vecinos y cuyo origen puede estar tanto en el proceso de adquisición de la imagen como de la transmisión.
- Realzar bordes: destacar los bordes que se localizan en una imagen.
- Detectar bordes: detectar los pixeles donde se produce un cambio brusco en la función de la intensidad.

Existen cuatro métodos fundamentales de procesado digital: transformación punto a punto, ecualización de histogramas, métodos en el dominio espacial y métodos en el dominio de la frecuencia.

2.2.3.1. Métodos de transformación punto a punto

Son transformaciones punto a punto las que se realizan sobre el rango de los niveles de gris. Son de fácil implementación y se pueden realizar en tiempo real, a frecuencia de digitalización. Se pueden expresar como una función

$$v = T(u)$$

donde u es el nivel de gris de la imagen original y v el transformado.

- **Dilatación del rango dinámico**

  Se aplica a imágenes pobremente contrastadas debido a una mala iluminación (aparecen muchos puntos en un intervalo pequeño de niveles de gris). Interesa resaltar la zona donde hay una mayor concentración de niveles de gris. En este caso el valor de la transformación viene dado por:

$$v = \begin{cases} \propto u & si\ 0 \le u < a \\ \beta(u-a) + v_a & si\ a\ \le u < b \\ \gamma(u-b) + v_b & si\ b \le u < L \end{cases}$$

  La figura 4 es la representación gráfica de la anterior ecuación.

La pendiente de la transformación es mayor que la unidad en el rango $[a, b]$ a dilatar. Los valores a y b pueden calcularse examinando el histograma[1] de la imagen.

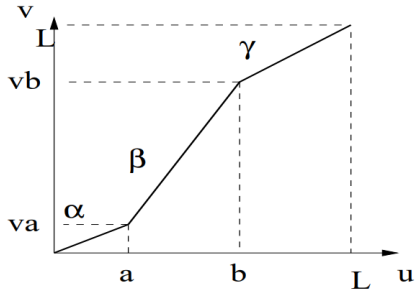

Figura 4. Pendiente de un rango dinámico.

- **Fraccionamiento del nivel de gris**

Es un caso especial de la dilatación de contraste cuando $\alpha = \gamma = 0$

Se conoce de antemano el rango $[a, b]$

$$v = \begin{cases} 0 & si\ 0 \leq u < a \\ \beta u & si\ a \leq u < b \\ L & si\ b \leq u < L \end{cases}$$

- **Negativo de la imagen**

Para obtener el negativo de una imagen basta con restar el rango del nivel de gris con el valor actual del pixel.

$$v = L - u$$

2.2.3.2. Filtrado en el dominio espacial

El término dominio espacial se refiere al conjunto de puntos que componen una imagen y los métodos en el dominio espacial son procedimientos que operan directamente sobre los pixeles. Las transformaciones de procesamiento de imágenes en el dominio espacial se pueden expresar como:

$$g(x, y) = T\{f(x, y)\}$$

[1] Un histograma es una gráfica donde se muestra el número de píxeles de cada nivel de gris.

donde $f(x,y)$ es la imagen de entrada, $g(x,y)$ es la imagen procesada, y T es el operador definido sobre alguna vecindad del punto (x,y). La principal técnica usada para definir una vecindad de (x,y) es usar una subimagen cuadrada centrada en (x,y).

El centro de la subimagen se mueve de punto a punto empezando en la esquina superior izquierda y al aplicar el operador en cada punto obtenemos el valor de la nueva imagen $g(x,y)$ en cada punto.

Una de las formas de realizar este proceso es mediante máscaras, también denominadas *kernel*. Básicamente una máscara es una matriz bidimensional pequeña (por ejemplo, $3x3$) cuyo valor de los elementos son escogidos para detectar una propiedad de la imagen.

w_1 $(x-1, y-1)$	w_2 $(x, y-1)$	w_3 $(x+1, y-1)$
w_4 $(x-1, y)$	w_5 (x, y)	w_6 $(x+1, y)$
w_7 $(x-1, y+1)$	w_8 $(x, y+1)$	w_9 $(x+1, y+1)$

El centro de la máscara se mueve a lo largo de la imagen. En cada posición multiplicamos cada punto que está contenido dentro del área que ocupa la máscara por el correspondiente coeficiente de la máscara. Entonces, el valor de cada punto de la nueva imagen $g(x,y)$ se obtiene como:

$$\begin{aligned} g(x,y) &= T\big(f(x,y)\big) \\ &= w_1 f(x-1,y-1) + w_2 f(x,y-1) + w_3 f(x+1,y-1) \\ &+ w_4 f(x-1,y) + w_5 f(x,y) + w_6 f(x+1,y) + w_7 f(x-1,y+1) \\ &+ w_8 f(x,y+1) + w_9 f(x+1,y+1) \end{aligned}$$

En la figura 5 se puede ver cómo se realiza el proceso.

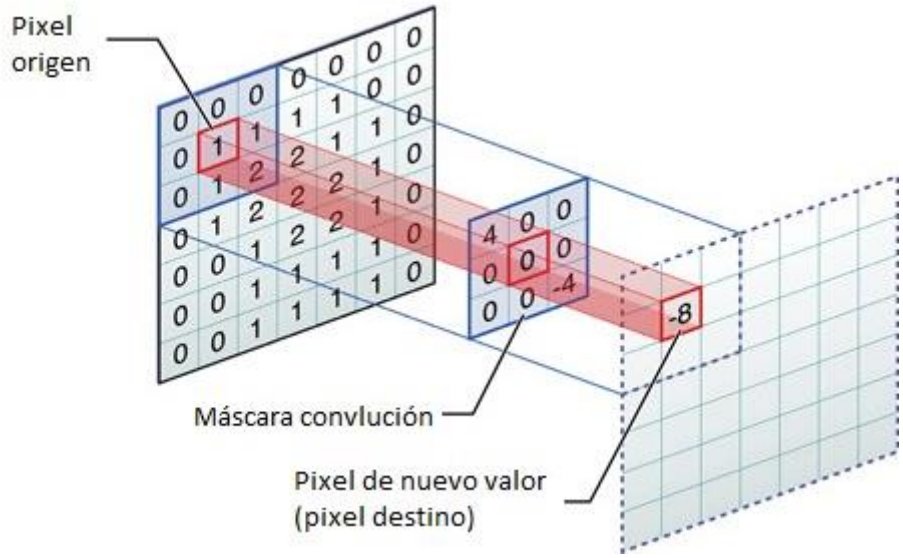

Figura 5. Aplicación de una máscara de convolución.

- **Suavizado de imágenes**
  Las operaciones de suavizado se utilizan para disminuir los efectos negativos que se pueden presentar en una imagen digital como consecuencia de un sistema de muestreo pobre o del canal de transmisión.
  - **Promedio de vecinos**

    Dada una imagen $f(x,y)$ de tamaño NxN, el valor del nivel de gris de la imagen suavizada $g(x,y)$ en el punto (x,y) se obtiene promediando los valores de nivel de gris de los puntos de f contenidos en una cierta vecindad de (x,y):

    $$g(x,y) = \frac{1}{M} \sum_{(n,m)\in S} f(n,m)$$

    donde $x,y = 0,1 \ldots., N-1$, S es el conjunto de coordenadas de los puntos vecinos a (x,y), incluyendo el propio (x,y), y M es el número de puntos de la vecindad.

    Esta operación es equivalente a aplicar un proceso de convolución con una máscara cuyos coeficientes son todos $1/M$, por ejemplo, para una máscara $3x3$, M sería 9 y todos los coeficientes de la máscara $1/9$.

- **Filtros de mediana**

  Uno de los principales problemas del método anterior es que difumina los bordes y otros detalles de contraste. Un método alternativo es utilizar filtros de mediana, en los que remplazamos el valor de gris de un punto por la mediana de los niveles de gris de una cierta vecindad.

  Recordemos que la mediana m de un conjunto de valores es aquel tal que la mitad de los valores del conjunto son menores que m y la otra mitad son mayores que m.

  La función principal de los filtros de mediana es forzar a los puntos con valores de intensidad muy distintos a sus vecinos a tener valores más próximos a sus vecinos, de modo que se eliminan los picos de intensidad que aparecen en áreas aisladas.

- **Técnicas basadas en gradiente. Método de detección de bordes.**

  Las técnicas basadas en gradiente usan la primera derivada de la imagen y buscan el máximo y el mínimo de esa derivada. El motivo de emplear la primera derivada es que toma el valor cero en todas las regiones donde no varía la intensidad y apunta en la dirección de la máxima variación (su módulo es proporcional a dicha variación). Por tanto, un cambio de intensidad se manifiesta como un cambio brusco en la primera derivada. Esta característica es usada para detectar un borde. Así, en el caso de funciones bidimensionales $f(x,y)$:

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix}$$

$$\nabla f(x,y) = \sqrt{\left(\frac{\partial f(x,y)}{\partial x}\right)^2 + \left(\frac{\partial f(x,y)}{\partial y}\right)^2}$$

  Para una imagen digital el gradiente de fila puede aproximarse por la diferencia de píxeles adyacentes de la misma fila:

$$G_x(i,j) = \frac{\partial f(x,y)}{\partial x} \approx f(x,y) - f(x-1,y) = [-1 \quad 1] \times \begin{bmatrix} f(x-1,y) \\ f(x,y) \end{bmatrix}$$

  De igual manera el gradiente de columna puede obtenerse mediante:

$$G_y(i,j) = \frac{\partial f(x,y)}{\partial y} \approx f(x,y) - f(x,y-1)$$
$$= [f(x,y-1) \quad f(x,y)] \times \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

El gradiente de fila G_x y de columna G_y en cada punto se obtienen mediante la convolución de la imagen con las máscaras H_x y H_y, esto es:

$$G_x(i,j) = F(i,j) \otimes H_x(i,j)$$
$$G_y(i,j) = F(i,j) \otimes H_y(i,j)$$

La magnitud puede aproximarse por:

$$|G(i,j)| = \sqrt{G_x^2 + G_y^2} \approx |G_x(i,j)| + |G_y(i,j)|$$

Esta técnica ha sido la utilizada para la detección de bordes de este proyecto, en el apartado 4.1 se explica cómo es implementada esta técnica, junto con el operador de Sobel.

- **Operador Sobel.**

  El operador de Sobel define el tamaño y los coeficientes de las máscaras H_x y H_y utilizadas en métodos basados en gradiente.

$$H_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad H_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

  Nótese que una máscara es el resultado de girar 90º la otra.

  La propiedad de este operador está en suavizar la imagen, eliminar parte del ruido.

  Estas máscaras obtienen su mejor rendimiento con bordes que tienen orientaciones verticales y horizontales.

  Existen otros operadores como el de Roberts Cross que proporciona mayor velocidad de computación, o el de Prewitt que da más peso al área de estimación.

  En este proyecto nos centramos en el uso exclusivo del operador Sobel.

2.2.3.3. Filtrado en el dominio de la frecuencia

Los filtros en el dominio de la frecuencia trabajan sobre la Transformada de Fourier de la imagen. Para ello, ésta se modifica siguiendo el Teorema de la Convolución correspondiente con los siguientes pasos:

- Aplicar Transformada de Fourier.
- Multiplicación del resultado por la función filtro deseada.
- Restaurar al dominio espacial empleando la Transformada Inversa de Fourier.

Por tanto:

$$G(u,v) = F(u,v) * H(u,v)$$

donde:

$F(u,v)$: transformada de Fourier de la imagen original

$H(u,v)$: filtro de frecuencias

A continuación, en la figura 6, se representan las etapas del procesado de imágenes:

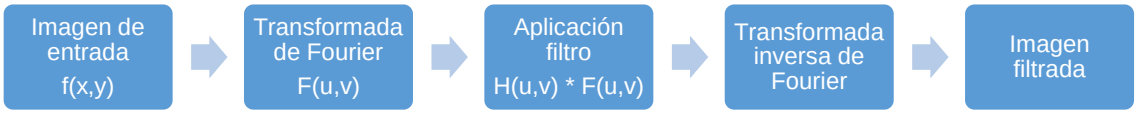

Figura 6. Etapas del procesamiento de imágenes en el dominio de la frecuencia.

- **Tipos de filtro**
  - **Filtro paso bajo**

    Atenúa las frecuencias altas manteniendo sin variaciones las bajas. El resultado en el dominio espacial es el equivalente al de un filtro de suavizado, donde las altas frecuencias que son filtradas se corresponden con los cambios fuertes de intensidad. Consigue reducir el ruido suavizando las transiciones existentes.

  - **Filtro paso alto**

    Atenúa las frecuencias bajas manteniendo invariables las frecuencias altas. Puesto que las altas frecuencias corresponden a cambios bruscos de densidad, este tipo de filtros es usado, porque entre otras ventajas, ofrece mejoras en la detección de bordes en

el dominio espacial, ya que estos contienen gran cantidad de dichas frecuencias. Refuerza los contrastes que se encuentran en la imagen.

- **Filtro paso banda**

Atenúa frecuencias muy altas o muy bajas manteniendo una banda de rango medio.

2.3. FPGA como mecanismo de procesado

Las FPGA (‘*Field Programable Gate Array*’) o matriz de puertas programables, son dispositivos de silicio prefabricados que pueden ser eléctricamente programados en campo para convertirse en casi cualquier tipo de circuito digital o sistema. Para bajos volúmenes de producción, las FPGA proporcionan una solución más barata y rápida comparada con las ASIC (‘*Application Specific Integrated Circuits*’ o Circuitos Integrados de Aplicación Específica) los cuales requieren de mucho tiempo de producción y dinero de inversión. La naturaleza flexible de las FPGA las hace más grandes, más lentas y menos eficientes que sus equivalentes ASIC. Estos inconvenientes son debidos a que la interconexión programable de las FPGA ocupa casi el 90% del área total, elemento que no existe en los ASIC (Farooq, 2012).

Normalmente las FPGA constan de:

- Bloques lógicos configurables (‘*Configurable Logic blocks*’, CLBs) que implementan funciones lógicas.
- Interconexiones programables que conectan esas funciones lógicas.
- Bloques de E/S que están conectados a los bloques lógicos mediante la interconexión programable y que permiten la conexión con el exterior del chip.

Un ejemplo generalizado de una FPGA se muestra en la figura 7, donde los bloques lógicos configurables (CLBs) se disponen en una cuadrícula bidimensional interconectada por la red de interconexión programable. Los bloques de E/S (en inglés, I/O) se disponen en la periferia de la cuadrícula, también conectada a la red de interconexión programable.

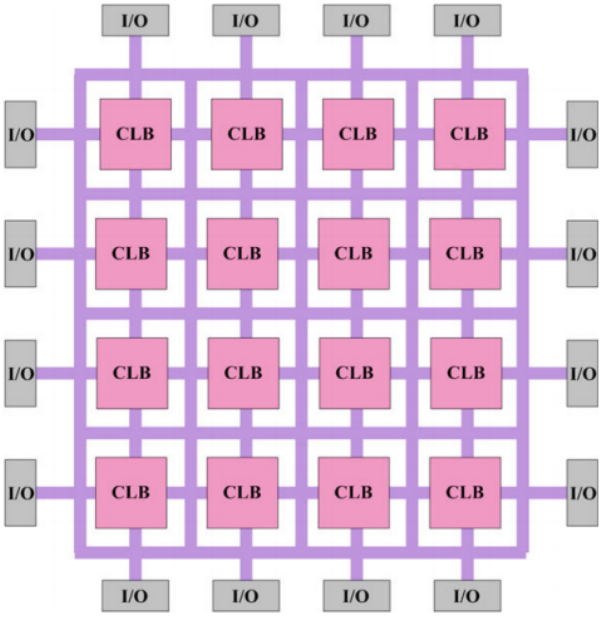

Figura 7. Visión general de la arquitectura de una FPGA.

2.3.1. Bloques lógicos configurables. CLBs.

Un bloque lógico configurable (CLB) es un componente básico de una FPGA que proporciona la lógica básica y funcionalidad de almacenamiento para el diseño de una aplicación específica.

Los principales fabricantes utilizan CLBs basados en LUT (*'lookup tables'*). Este tipo de CLBs proporcionan un buen equilibrio entre bloques demasiado extensos y bloques demasiado complejos para pequeñas funciones.

Un CLB puede estar compuesto de un solo elemento básico lógico (*'basic logic element'* BLE) o un grupo de ellos interconectados localmente.

Un BLE consiste en un LUT y un Flip-Flop. Un LUT con k entradas (LUT-k) contiene 2^k bits de configuración y puede implementar cualquier función de k entradas. En la figura 8 se muestra un LUT de 4 entradas (LUT-4) y un Flip-Flop de tipo D. El LUT-4 utiliza 16 bits de programación para implementar cualquier función de 4 bits de entrada. La salida del LUT-4 está conectada a un Flip-Flop opcional, finalmente un multiplexor selecciona la salida del BLE para ser la salida del Flip-Flop o la del LUT-4.

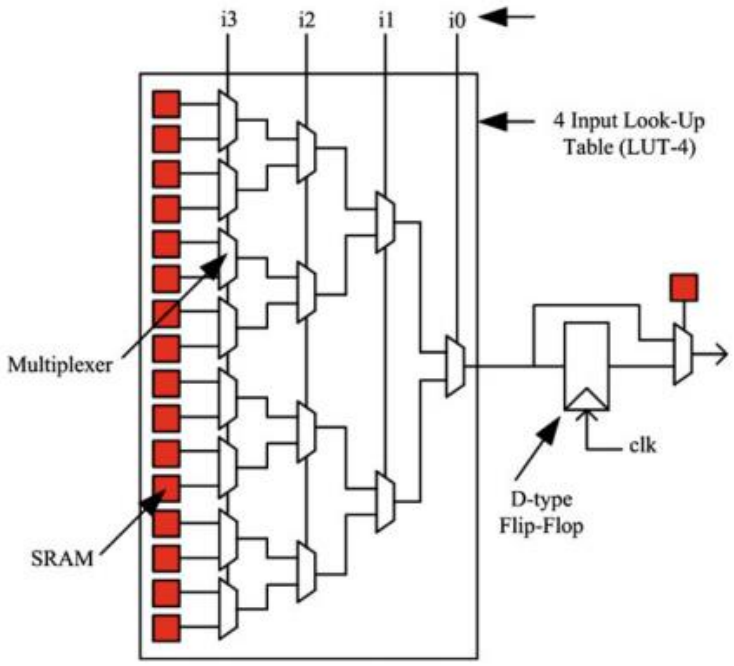

Figura 8. Bloque lógico básico (BLE).

Un CLB puede contener un grupo de BLEs conectados localmente entre sí. Por ejemplo, en una FPGA con arquitectura Spartan-6 de Xilinx (Xilinx, 2011), utilizada en este trabajo, un CLB está compuesto por dos Slices. Un slice, denominado por Xilinx, es un bloque lógico de menor rango que un CLB por el que estos están compuestos. En la figura 9 podemos ver la disposición de los dos slices en un CLB y el acceso de ellos a la red de interconexión. Nótese la salida y entrada de acarreo del Slice(0), esta va dirigida al CLB superior e inferior, respectivamente, dentro de la cuadrícula.

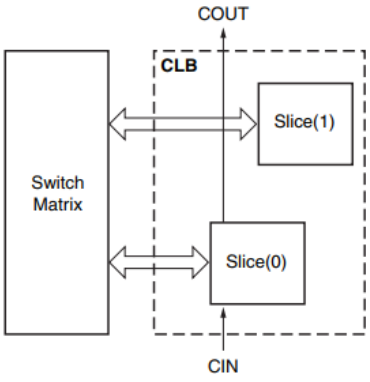

Figura 9. CLB de una FPGA Spartan-6 de Xilinx.

Continuando con la arquitectura de una FPGA Spartan-6 (Xilinx, 2010), existen tres tipos de configuraciones de Slices: SLICEX, SLICEL y SLICEM. En

la tabla 1 se pueden ver las características de cada uno de ellos. Uno de los dos slices de un CLB es un SLICEX mientras que el otro se alterna entre SLICEL y SLICEM.

Característica	SLICEX	SLICEL	SLICEM
LUT-6	✓	✓	✓
8 Flip-Flops	✓	✓	✓
Multiplexores anchos		✓	✓
Lógica de acarreo		✓	✓
RAM distribuida			✓
Registros desplazamiento			✓

Tabla 1. Comparación entre slices.

2.3.2. Arquitecturas de interconexión.

Cómo se ha comentado anteriormente, la funcionalidad de computación es proporcionada tanto por los bloques lógicos programables como por la red programable que conecta esos bloques y los bloques de E/S.

La red de interconexión de una FPGA consiste en cables y conmutadores programables. Estos conmutadores se programan igual que si se tratara de un LUT, basados en la misma tecnología.

Existen notables diferencias entre las arquitecturas de interconexión de cada fabricante, incluso dentro de diferentes familias de estos, pero a grandes rasgos se pueden diferenciar dos tipos de arquitecturas: estilo isla y jerárquica.

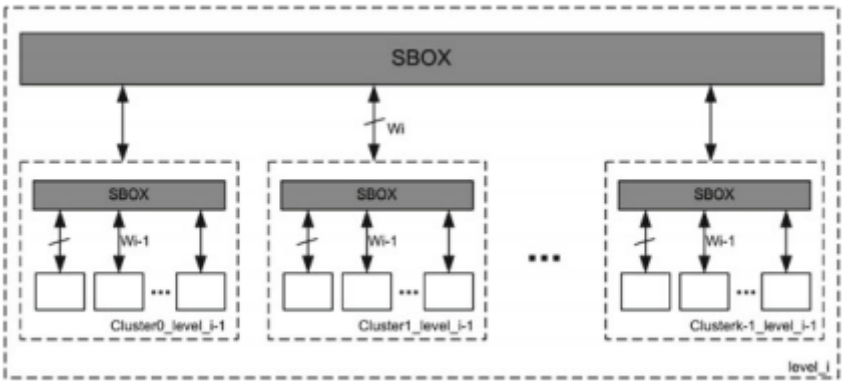

Figura 10. Arquitectura jerárquica de una red de interconexión.

Una arquitectura jerárquica divide distintos bloques lógicos en grupos separados, de manera que existen conexiones locales entre bloques del mismo nivel, pero para acceder a bloques alejados necesitan recorrer la red troncal de interconexiones, como si se tratara de un árbol. En la figura 10 podemos ver un ejemplo de arquitectura jerárquica típico.

La arquitectura de interconexión de una FPGA Spartan-6 es de tipo isla, por lo que la utilizaremos para explicar este tipo de arquitectura. (Xilinx, 2010)

Los CLBs se disponen en una matriz. Cada uno se conecta a una matriz de conmutadores que le dan acceso a la red de interconexiones que recorre la FPGA entre los CLBs en filas y columnas. Es esta última matriz la que es programable y permite todo tipo de interconexiones entre CLBs. En la figura 11 podemos ver claramente en que consiste esta arquitectura.

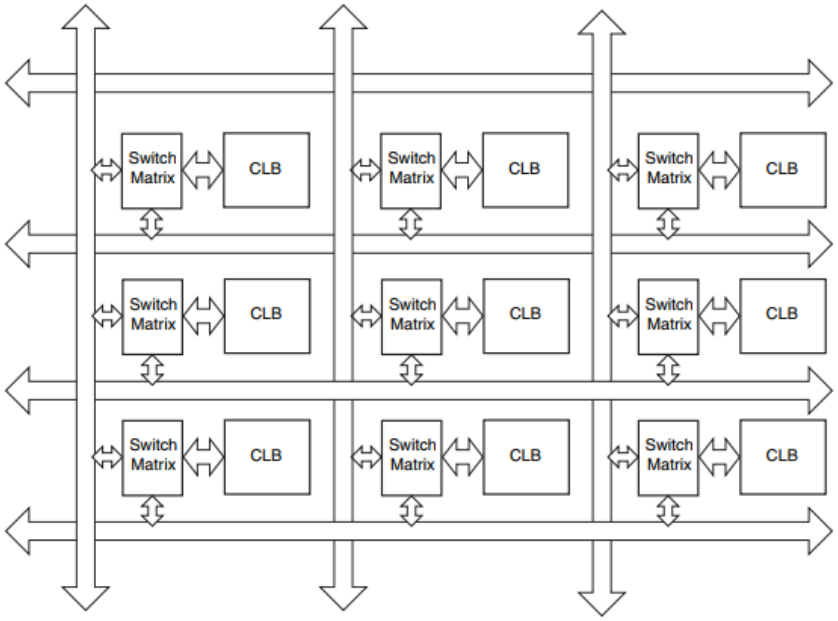

Figura 11. Arquitectura interconexión de una FPGA Spartan-6.

2.3.3. Bloques especiales.

Cada fabricante de FPGA añade una serie de bloques especiales que resultan bastante variados entre ellos, cometiendo una función muy específica. Estos bloques proporcionan una gran eficiencia para la tarea para la que han sido desarrollados, pero por otro lado suelen ocupar mucho espacio en planta.

Los bloques especiales que nos proporciona la familia Spartan-6 son los siguientes:

- Administración de relojes:
  - DCM
  - PLL
- Distribución de relojes:
  - Global Clock Lines
  - I/O Clocks
- Bloques RAM.
- Bloques de control de memoria (‘*Memory Controller Block’*, MCB)
- Bloques DSP (‘Digital Signal Processing’ – DSP48A1 Slice)
- Gigabit Transceiver
- PCI Express

2.4. Xilinx ISE

Xilinx ISE (Integrated Synthesis Environment) es un conjunto de herramientas software elaborada por Xilinx destinada a analizar y sintetizar diseños HDL, permitiendo al desarrollador además realizar análisis de temporización, examinar diagramas RTL, simular reacciones de un diseño en base a diferentes estímulos y configurar el dispositivo elegido con el programador (Mendías, 2015).

Es un entorno de desarrollo orientado a FPGA y CPLDs exclusivamente de Xilinx, por lo que está altamente optimizado para el uso con estas.

Xilinx ISE se usa principalmente para la síntesis de circuitos, mientras que para el testeo a nivel del sistema permite el uso de ISIM o ModelSim. Las simulaciones pueden ser funcionales o *post-route*. Las simulaciones funcionales recrean el funcionamiento del código VHDL en base a unos estímulos, mientras que las simulaciones *post-route* además tienen en cuenta características reales de cada FPGA, por lo que añaden comportamientos reales como retrasos en las señales y frecuencias máximas soportadas de funcionamiento.

Herramientas incluidas en el entorno:

- **Project Navigator**: Interfaz gráfico común de las herramientas EDA de Xilinx. Incluye la gestión de proyectos y los algoritmos de síntesis e implementación.
- **Plan Ahead**: Interfaz gráfico alternativo al *Project Navigator*.
- **iSIM Simulator**: Simulador HDL para la simulación funcional y temporal de diseños.
- **Core Genrator**: Generador de componentes virtuales (*Intellectual Property cores*).
- **Schematic Viewer**: Visualizador del esquemático de un diseño.
- **Timing Analyzer**: Analizador estático de tiempos.
- **FPGA Editor**: Editor físico para el emplazamiento y rutado manual de un diseño.
- **XPower Analyzer**: Analizador estático de consumo.
- **iMPact**: Permite el volcado y readback de bitstreams.
- **ChipScope**: Permite el análisis lógico en tiempo real de un diseño.
- **Constraints Editor**: Interfaz gráfico para la edición de constraints de diseño e implementación.

A continuación, se muestran dos capturas del entorno Xilinx ISE y del simulador ISim.

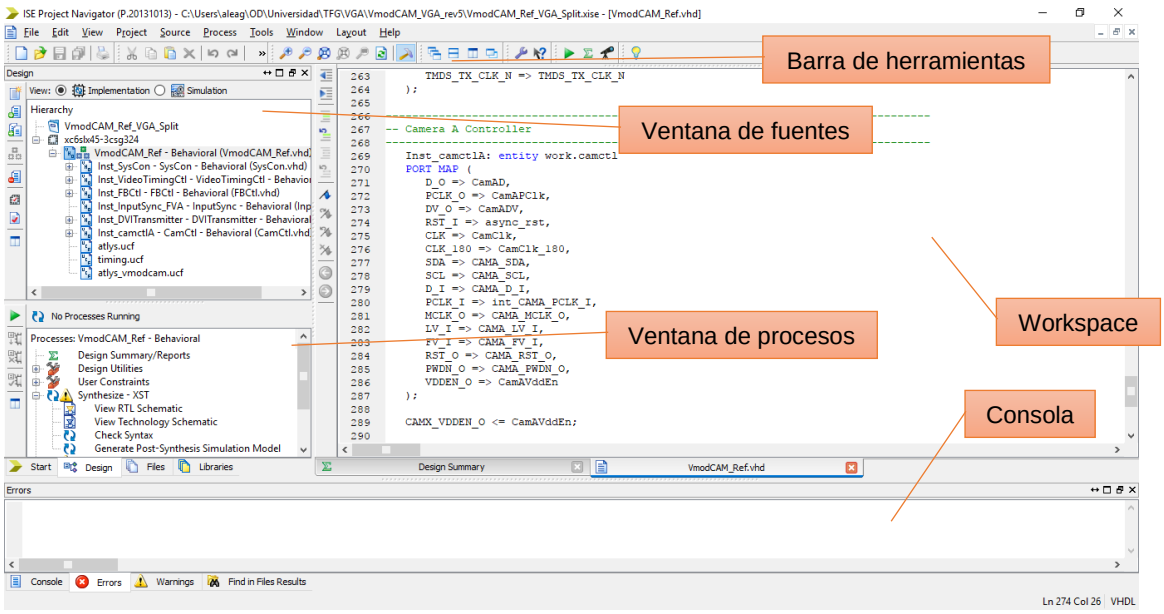

Figura 12. Captura de un proyecto de Xilinx ISE.

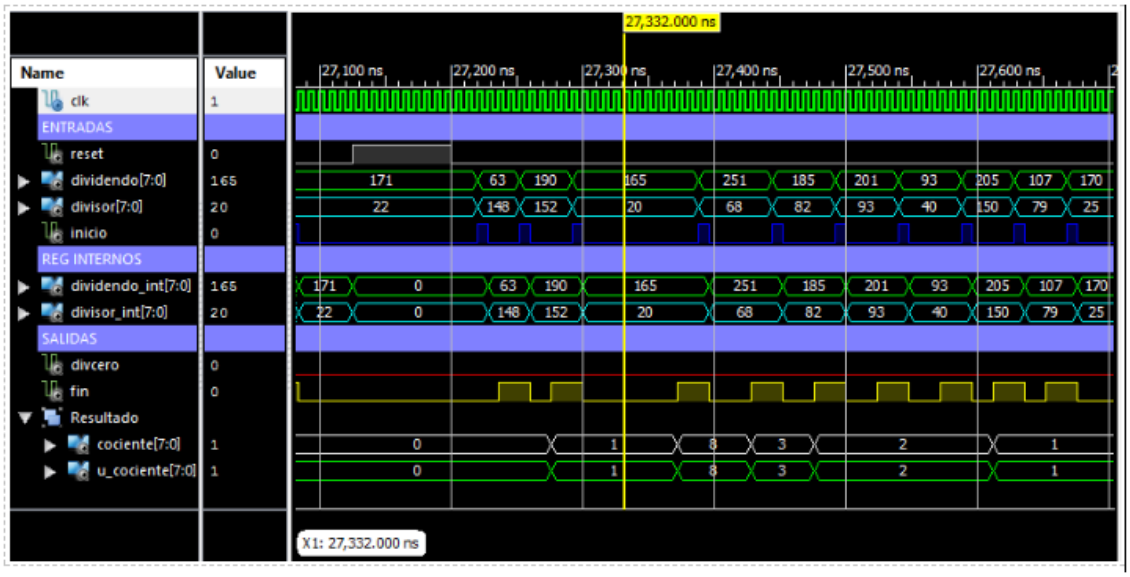

Figura 13. Simulación mediante Isim.

2.4.1. Metodología de diseño

La metodología de diseño, o flujo de diseño, se puede dividir en tres grandes etapas en las que iremos de más alto nivel a más bajo: diseño VHDL, síntesis del diseño e implementación del diseño. Cada etapa lleva asociada una simulación por software en la cual cada vez se introducen más restricciones, lo que provoca un comportamiento más cercano al real, a su vez permitiendo descubrir posibles errores de diseño.

- Diseño VHDL:

En esta primera etapa, el diseñador comienza el proyecto, por lo que debe plantear introducir el sistema necesario mediante la realización de esquemas y diagramas de cómo va a abordar el problema. El procedimiento que se suele utilizar es una descomposición jerárquica Arriba-Abajo (*Top-Down*), estableciendo los bloques necesarios y las señales que los conectan. A su vez, estos bloques son descompuestos en otros más simples, de menor rango en la jerarquía.

A continuación, se comienza a diseñar los bloques de menor rango jerárquico. Estos deben ser verificados mediante simulación, utilizando un banco de pruebas (*testbench*) específico para ese bloque. Una vez terminado un nivel de jerarquía, se procede a diseñar los bloques inmediatamente superiores que dependen de otros bloques ya diseñados, también mediante la utilización de *testbenchs*. A esta fase de la metodología se la denomina Abajo-Arriba (Bottom-Up).

Los bloques diseñados suelen incorporarse a una librería de diseño para su reutilización a lo largo del proyecto. Este concepto se denomina reusabilidad.

Toda esta etapa es independiente de la tecnología o fabricante elegido, por lo que es posible utilizar cualquier entorno de desarrollo VHDL y simulador asociado.

- Síntesis del diseño:

Eesta etapa se centra en el diseño una FPGA específica de un fabricante y su respectivo entorno de diseño, en este caso Xilinx ISE.

Al depender de una FPGA concreta podemos añadir bloques funcionales específicos, como por ejemplo bloques de control DDR, o IP cores proporcionados por Xilinx.

En esta etapa se realiza la síntesis del diseño, en la que mediante el entorno de diseño se realiza una primera implementación en la FPGA utilizando los recursos disponibles y las conexiones necesarias o netlist. De esta síntesis se generan informes proporcionando datos del porcentaje de recursos utilizados, por lo que es interesante a la hora de elegir el modelo comercial concreto de FPGA.

La síntesis del diseño lleva asociada la simulación funcional. En teoría esta simulación es parecida a la realizada en la etapa anterior, pero nos garantiza una mejor aproximación a la realidad.

- Implementación del diseño:

En esta última etapa se añaden restricciones temporales, se podría decir que recrea casi a la perfección la realidad. La etapa anterior simula el funcionamiento de la FPGA de forma ideal, pero en este punto se tienen en cuenta los retrasos de las señales debido a la propagación y la consecuente desincronización de señales.

Por ejemplo, podemos estar diseñando un sistema en el que se utilizan altas frecuencias. Aunque en la simulación de la etapa anterior funcione correctamente, cuando realicemos la simulación de esta etapa o simulación temporal, detectemos que el diseño no funciona correctamente, por lo que requeriría de un rediseño del sistema.

En la figura 14 podemos ver un diagrama de las etapas del entorno Xilinx ISE descritas anteriormente.

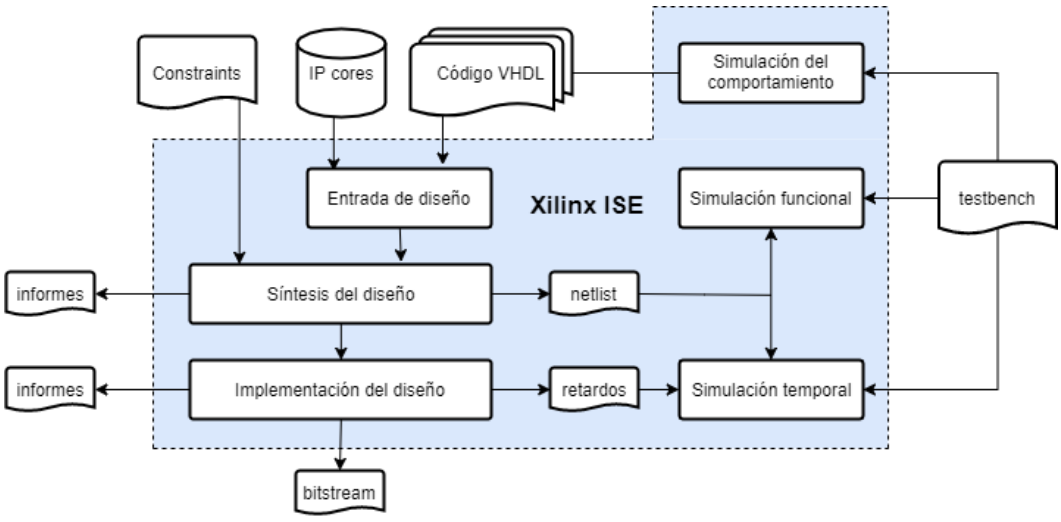

Figura 14. Diagrama de la metodología de diseño.

2.5. Lenguaje de descripción de hardware. VHDL.

VHDL es un lenguaje de especificación definido por el IEEE (Institute of Electrical and Electronics Engineers) (ANSI/IEEE 1076-1993) utilizado para describir circuitos digitales y para la automatización de diseño electrónico.

VHDL es un acrónimo proveniente de la combinación de otros dos: VHSIC (Very High Speed Integrated Circuit) y HDL (Hardware Description Language). Aunque puede ser usado de forma general para describir cualquier circuito digital se usa principalmente para programar PLD (Programable Logic Device - Dispositivo Lógico Programable), FPGA (Field Programmable Gate Array), ASIC y similares.

Otros métodos para diseñar circuitos son la captura de esquemas (con herramientas CAD) y los diagramas de bloques, pero estos no son prácticos en diseños complejos. Otros lenguajes para el mismo propósito, pero con un nivel de abstracción superior son Verilog y ABEL.

VHDL es un lenguaje que permite tanto modelar un circuito de real implementación, como realizar la simulación del mismo, para ello se describe un banco de pruebas, donde se van estimulando las entradas del circuito y se revisa la respuesta a ellos. Cabe destacar que las restricciones del lenguaje en simulación son nulas, mientras que en VHDL “sintetizable” algunas sentencias están prohibidas.

2.5.1. Elementos básicos de VHDL

Un sistema digital está descrito por sus entradas y sus salidas y la relación que existe entre ellas.

En el caso de VHDL por un lado se describirá el aspecto exterior del circuito: entradas y salidas; y por otro la forma de relacionar las salidas con las entradas. El aspecto exterior, cuántos puertos de entrada y salida tenemos, es lo que se denomina **entity**. Y la descripción del comportamiento del circuito **architecture**, toda *architecture* tiene que estar asociada a una *entity*.

Además, aunque no es estrictamente necesario, podemos definir también las bibliotecas y paquetes que vamos a utilizar, lo que nos indicará que tipos de puertos y operadores podemos utilizar. Siempre ha de aparecer la definición de las bibliotecas y paquetes antes de la definición de la *entity*.

2.5.1.1. Entity

Una entidad es la abstracción de un circuito, ya sea desde un complejo sistema electrónico o una simple puerta lógica. La entidad únicamente describe la forma externa del circuito, en ella se enumeran las entradas y las salidas del diseño. Una entidad es análoga a un símbolo esquemático en los diagramas electrónicos, el cual describe las conexiones del dispositivo hacia el resto del diseño.

- Define externamente al circuito o subcircuito.
- Nombre y número de puertos, tipos de datos de entrada y salida.
    - Puertos: interconexiones de las señales internas con el exterior. Tipos:
        - in: entrada.
        - out: salida
        - inout: bidireccional.
        - buffer: salida del circuito de sólo lectura.
    - Generics: se utilizan para declarar propiedades y constantes del circuito en el momento de la instanciación.
- Tiene toda la información necesaria para conectar elcircuito a otros circuitos.

```
entity nombre is
    generic (cte1: tipo := valor1; cte2: tipo := valor2;…);
    port (entrada1, entrada2,…: in tipo;
          salida1, salida2, …: out tipo;
          puertoi : modo tipo);
end nombre;
```

2.5.1.2. Architecture

Los pares de entidades y arquitecturas se utilizan para representar la descripción completa de un diseño. Una arquitectura describe el funcionamiento de la entidad a la que hace referencia, es decir, dentro de **architecture** tendremos que describir el funcionamiento de la entidad a la que está asociada utilizando las sentencias y expresiones propias de VHDL.

- Define internamente el circuito.
- Señales internas, funciones, procedimientos, constantes …
- La descripción de la arquitectura puede ser estructural o por comportamiento.

```
architecture arch_name of entity_name is
-- Declaraciones de la arquitectura:
-- tipos
-- señales
-- componentes
begin
-- Código de descripción
-- instrucciones concurrentes
-- ecuaciones booleanas
-- componentes
        -- process (lista de sensibilidad)
        -- begin
        -- código de descripción
        end process;
end arch_name;
```

El código VHDL propiamente dicho se escribe dentro de *architecture*. Cada *architecture* va asociada a una *entity* y se indica en la primera sentencia. A continuación, y antes de *begin* se definen todas las variables (señales) internas que se necesitan para describir el comportamiento de nuestro circuito, se definen los tipos particulares que hay que utilizar y los componentes, otros circuitos ya definidos y compilados de los cuales conocemos su interfaz en VHDL (su *entity*).

Desde **begin** hasta **end** escribiremos todas las sentencias propias de VHDL, pero no todas pueden utilizarse en cualquier parte del código. Así pues, aquellas sentencias de VHDL que tengan definido un valor para cualquier valor de la entrada (sentencias concurrentes) podrán ir en cualquier parte del código, pero fuera de la estructura *process*. Aunque no es el fin de este manual, puede afirmarse que todas las sentencias concurrentes se traducirán en subcircuitos combinacionales. También fuera de la estructura *process*, se instanciarán los componentes, subcircuitos ya definidos utilizados por el circuito actual, indicando cuáles son sus entradas y sus salidas de entre las señales del circuito del que forman parte.

El **process** es una estructura particular de VHDL (que se describe con mucho más detalle más adelante) que se reserva principalmente para contener sentencias que no tengan obligatoriamente que tener definido su valor para todas las entradas (el ejemplo más común es una estructura *if-else* incompleta). Esto obliga a que la estructura *process* almacene los valores de sus señales y pueda dar lugar (no siempre) a subcircuitos secuenciales. Además, en simulación sólo se ejecutan las sentencias internas a esta

estructura cuando alguna de las señales de su lista de sensibilidad cambia de valor.

2.5.1.3. Identificadores

En VHDL existen tres clases de objetos por defecto:

- **Constant**. Los objetos de esta clase tienen un valor inicial que es asignado de forma previa a la simulación y que no puede ser modificado durante ésta.
  - **constant** identificador: tipo:= valor;

- **Variable.** Los objetos de esta clase contienen un único valor que puede ser cambiado durante la simulación con una sentencia de asignación. Las variables generalmente se utilizan como índices, principalmente en instrucciones de bucle, o para tomar valores que permitan modelar componentes. Las variables no representan conexiones o estados de memoria. Pueden ser declaradas antes del *begin* de la *architecture* y/o antes del *begin* del *process*, en su declaración se les puede asignar un valor por defecto.
  - **variable** identificador: tipo [:= valor];

- **Signal.** Las señales representan elementos de memoria o conexiones y sí pueden ser sintetizados, dicho de otra manera, a cada objeto de nuestro código VHDL que sea declarado como *signal* le corresponde un cable o un elemento de memoria (biestable, registro …) en nuestro circuito. Por lo tanto, su comportamiento en simulación será el esperado de ese elemento físico, aunque no lo describamos en el código explícitamente. Tienen que ser declaradas antes del *begin* de la *architecture*. Los puertos de una entidad son implícitamente declarados como señales en el momento de la declaración, ya que estos representan conexiones.
  - **signal** identificador: tipo;

- **Tipos**. Formato de la constante, variable o señal. A continuación, se listan los tipos más comunes.
  - *Bit*
  - *Bit_vector*
  - *Boolean*
  - *Carácter*

- *String*
- *Integer rango*
- *Natural rango*
- *Positive rango*
- *Real rango*
- *Std_logic*
- *Std_logic_vector(rango)*
- *Alias*
- *Type*
- *Array*
- *Record*

2.5.1.4. Operadores

Un operador nos permite construir diferentes tipos de expresiones mediante las cuales podemos calcular datos utilizando diferentes señales. En VHDL existen distintos operadores de asignación con lo que se transfieren y transforman valores de una señal a otra.

Es importante recordar que para poder utilizar el tipo std_logic hay que añadir la librería que lo soporta:

use ieee.std_logic_1164.all

Para poder utilizar las funciones aritmeticológicas definidas (suma, resta, multiplcación) hay que añadir la librería:

use ieee.std_logic_arith.all

- Operaciones aritméticas: +,-,*,/,mod,rem
- Cambio de signo: +,-
- Concatenación: &
- Operaciones lógicas: and, or, nand, nor, xor
- Asignación de valores a constantes y variables: :=
- Asignación de valores a señales: <=

2.5.2. Estructura básica de un archivo fuente en VHDL

Como hemos visto, los modelos VHDL están formados por dos partes: la entidad (*entity*) y la arquitectura (*architecture*); es en esta última donde se escriben las sentencias que describen el comportamiento del circuito, a este modelo de programación en VHDL se le denomina *behavioral*.

architecture circuito **of** nombre **is**
-- señales
begin (lista de sensibilidad)
-- sentencias concurrentes
process (lista de sensibilidad)
begin
-- sentencias secuenciales
-- sentencias condicionales
end process
end architecture circuito;

Dentro de la arquitectura se encuentra:

i. Tipos y señales intermedias necesarias para la descripción del comportamiento.
ii. Sentencias de asignación que deben realizarse siempre, así como sentencias concurrentes.
iii. Uno o varios *process* que tienen en su interior sentencias condicionales y/o asignaciones a señales que dan lugar a hardware secuencial.

2.5.2.1. Sentencias concurrentes

Las sentencias concurrentes son sentencias condicionales que tienen al menos un valor por defecto para cuando no se cumplen ninguna de las condiciones. Aunque podría utilizarse una sentencia común como un *if* con obligación de *else*, los desarrolladores de VHDL han preferido utilizar dos sentencias particulares:

- WHEN-ELSE
- WITH-SELECT-WHEN

2.5.2.2. Sentencias condicionales

VHDL permite utilizar otro tipo de sentencias condicionales más parecidas a los lenguajes de programación habituales. Todas estas sentencias como se explicará en la siguiente sección, tienen que ir obligatoriamente dentro de un *process*. Las sentencias condicionales más comunes en VHDL son las siguientes:

- IF-THEN-ELSE
- CASE-WHEN
- FOR-LOOP
- WHILE-LOOP

2.5.2.3. Sentencia process

VHDL presenta una estructura particular denominada *process* que define los límites de un dominio que se ejecutará si y sólo si alguna de las señales de su lista de sensibilidad se ha modificado en el anterior paso. Un *process* tiene la siguiente estructura.

```
process (lista_de_sensibilidad)
-- asignación de variables
-- opcional no recomendable
begin
-- sentencias condicionales
-- asignaciones
End process;
```

La sentencia *process* es una de las más utilizadas en VHDL ya que tanto las sentencias condicionales como la descripción de hardware secuencial se realiza dentro de ella.

2.5.2.4. Descripción estructural

Esta descripción utiliza para la creación de la arquitectura de la entidad, entidades descritas y compiladas previamente, de esta manera en VHDL podemos aprovechar diseños ya realizados, o realizar diseños sabiendo que se utilizarán en otros más complicados.

Se declaran los componentes que se van a utilizar y después, mediante los nombres de los nodos, se realizan las conexiones entre los puertos. Las descripciones estructurales son útiles cuando se trata de diseños jerárquicos *Bottom-up*.

```
architecture circuito of nombre is
    component subcircuito
    port(…);
    end component;
    --señales
begin
    chip_i: subcircuito port map (…);
    --se puede combinar con la descripción
    --behavioral (por comportamiento)
end circuito;
```

Se pueden utilizar tantos componentes como necesite el diseño, estos componentes son entidades anteriormente declaradas, en el ejemplo, subcircuito debe ser el nombre de una entidad que se ha declarado anteriormente en el código o que se ha compilado como parte de otro fichero

VHDL. Se puede utilizar un componente tantas veces como sea necesario (podemos asignar *chip_i* y *chip_j* al mismo componente, pero con distintas señales de entrada y salida).

Podemos realizar un diseño estructural puro, mediante la unión de componentes que describen las distintas funcionalidades del circuito, en el que necesitaremos definir las señales intermedias entre las que se encontrarán siempre las señales que interconectan las salidas de los módulos con las salidas de la *entity*.

3. SISTEMA DE DETECCIÓN DE BORDES. CARACTERÍSTICAS Y COMPONENTES.

En este capítulo se presenta el sistema utilizado, tanto la placa de desarrollo, la cámara, el diseño de demostración utilizado como base del proyecto. También se explica el funcionamiento de las interfaces complejas necesarias para el sistema.

Las interfaces presentadas son:

- Interfaz con la cámara: gestión y flujo de datos.
- Interfaz con memoria DDR.
- Interfaz HDMI.

3.1. Placa de desarrollo

Para este trabajo se ha utilizado la placa de desarrollo Atlys™ de Digilent® (figura 15) que incorpora principalmente una FPGA Xilinx Spartan-6 LX45 (XC6SLX45CSG324C) (Digilent, 2016). Las principales características de esta FPGA son:

- 6.822 slices, cada uno con seis LUT de entrada y ocho *flip-flops*.
- 2,1 Mbits de bloques de RAM rápida.
- Cuatro celdas de relojes (ocho DCM y cuatro PLL)
- Seis PLL.
- 58 DSP slices
- Velocidad de reloj mayor a 500 MHz.

Periféricos incluidos en la placa:

- 1 Gbit DDR2.
- 10/100/1000 Ethernet PHY.
- Dos puertos USB (para programación y transmisión de datos)
- Puerto USB para UART y ratón o teclado.
- Dos entradas y dos salidas HDMI.
- Codec AC-97 con puertos de entrada, salida, micrófono y auriculares.
- Monitorización en tiempo real de todas las alimentaciones.
- Memoria Flash SPI de 16 Mbytex4 para configuración y almacenamiento.
- Oscilador CMOS de 100 MHz.
- 48 entradas/salidas accesibles por conectores de expansión.
- Ocho LEDS, seis botones y ocho interruptores.

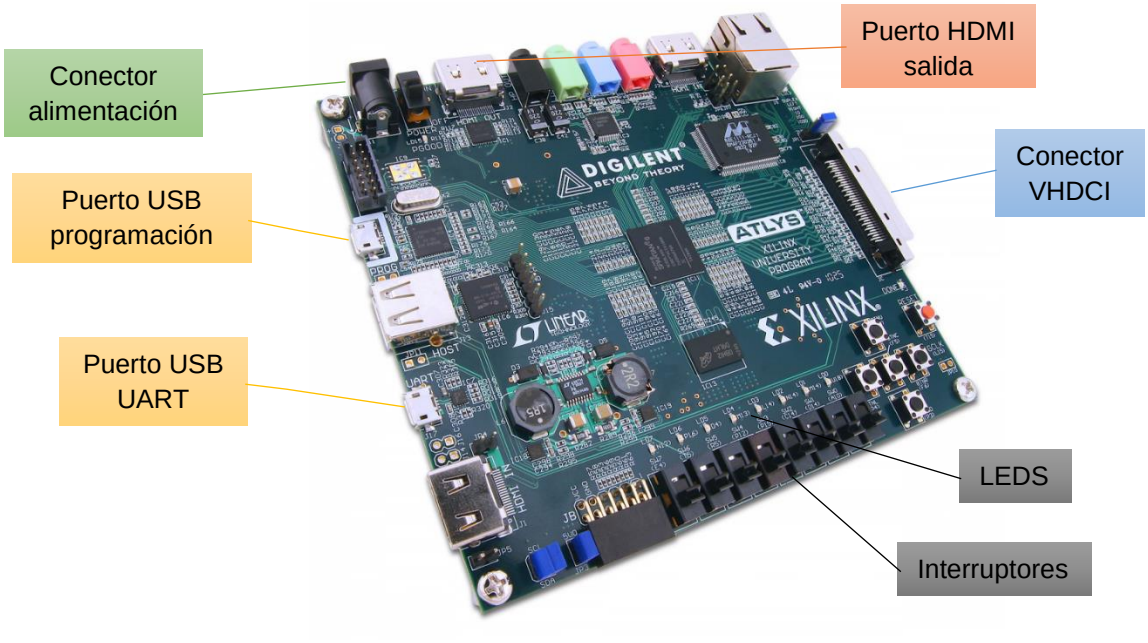

Figura 15. Placa de desarrollo Digilent Atlys.

La placa de desarrollo Atlys está diseñada especialmente para aplicaciones de video ya que cuenta con varios puertos HDMI de entrada y salida, suficiente memoria DDR2 y conector VHDCI de alta velocidad. Es por estos componentes por lo que ha sido idónea para este trabajo.

3.1.1. Memoria DDR2

Esta placa lleva un único chip de memoria DDR2, (MIRA P3R1GE3EGF G8E DDR2) el cual está preconfigurado para esta placa (Deutron Electronics Corp.). El fabricante nos informa de que la equivalencia comercial es P3R1GE3JGF.

Dispone de una densidad de 1Gbit, organizado en 8 bancos de 16MBytes y 1KB por página.

Su velocidad máxima de transferencia de datos es de 800 Mbps, con un ancho de banda de 8 bits.

Respecto al layout de la PCB, todas las señales de direccionamiento, datos, relojes y control tienen los retrasos emparejados (‘delay-matched’) e impedancia controlada. Además, las señales de control y direccionamiento están conectadas a 0,9V a través de resistencias de 47 ohms, las señales de datos utilizan la característica ‘On-Die-Termination’ (ODT) de este chip DDR2 y dispone de dos pares de relojes bien emparejados por lo que dispone de bajas desviaciones de reloj (‘low-skew clock’) desde la FPGA.

En la figura 16 podemos ver el conexionado entre la FPGA y la memoria DDR.

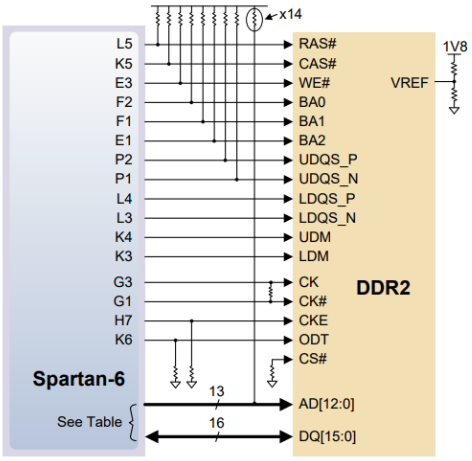

Figura 16. Esquema de conexión con la memoria DDR.

3.1.2. Memoria Flash

La placa Atlys lleva una memoria Flash SPI Numonyx N25Q12 de 128 Mbit, organizada en 16 Mbytes, para almacenamiento no volátil.

Puede ser programada con un fichero de configuración FPGA (‘.bit’, ‘.bin’ o ‘.mcs’) de menos de 12Mbits dejando 116Mbits disponibles para datos de usuario.

Los datos pueden ser transferidos bidireccionalmente por aplicaciones de usuario o mediante el sistema Adept integrado (figura 17).

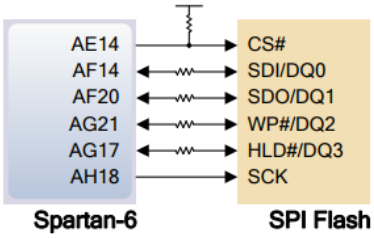

Figura 17. Esquema de conexión con la memoria Flash.

3.1.3. Puertos HDMI

Esta placa dispone de cuatro puertos HDMI, de los cuales dos puertos de entrada/salida y uno de salida disponen de buffer, y un último puerto que puede ser de entrada o salida no dispone de buffer.

Los puertos que disponen de buffer utilizan un conector HDMI A, mientras que el otro utiliza un conector HDMI D (figura 18).

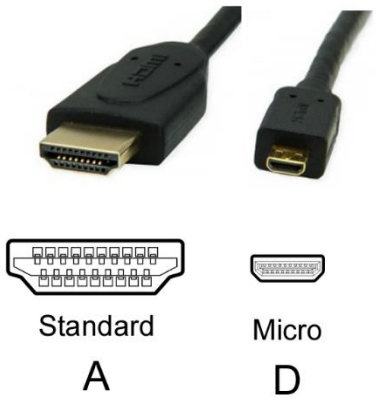

Figura 18. Tipos de puertos HDMI en la placa.

Los conectores HDMI de 19 pines incluyen cuatro canales de datos diferenciales, 5 conexiones de tierra, bus CEC (‘Consumer Electronics Control’) y un bus i2C denominado DDC (‘Display Data Channel’) (figura 19).

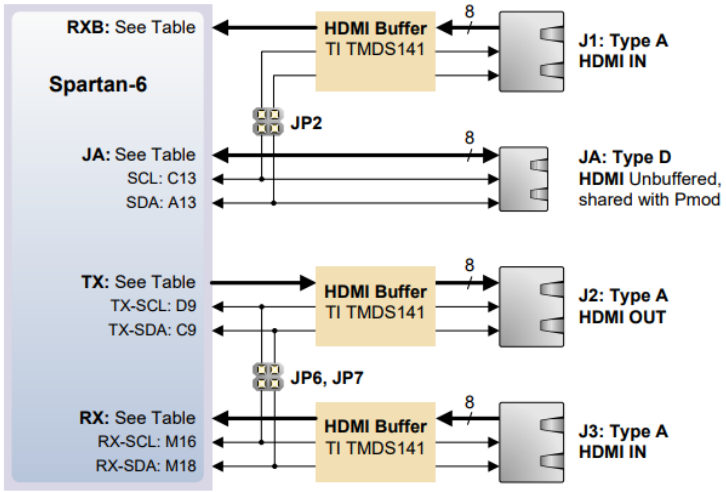

Figura 19. Esquemas de conexiones de los puertos HDMI.

3.1.4. Puente USB-UART

Para la transmisión de datos a través del puerto USB, la placa dispone del chip integrado EXAR XR21V1410 (figura 20) que es un Transmisor-Receptor Asíncrono Universal (UART) con interfaz USB, el cual soporta transferencias de hasta 12 Mbps. Dispone de una FIFO de 128 bytes de transmisión y otro de 384 bytes de recepción.

Soporta características UART como:

- 7, 8 o 9 bits de datos.
- 1 o 2 bits de stop.
- Control de flujo Hardware y Software (Xon/Xoff)

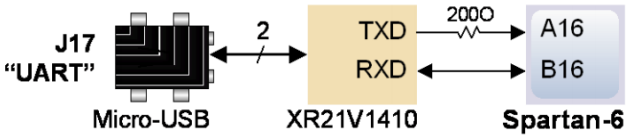

Figura 20. Esquema de conexión de la UART.

3.1.5. Entradas y salidas básicas

Cómo se ha mencionado anteriormente, la placa Atlys dispone de seis botones, ocho interruptores y ocho leds.

Los botones e interruptores están conectados a la FPGA mediante resistencias para prevenir daños por cortocircuitos. Todos los leds están conectados directamente a pines de la FPGA a través de resistencias de 390 ohms (figura 21), ya que estos leds de alta eficiencia se iluminan con una corriente de aproximadamente 1 mA cuando es aplicado un nivel lógico alto a su respectivo pin.

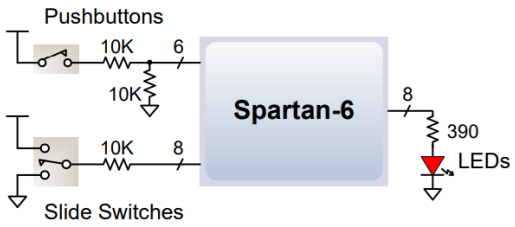

Figura 21. Esquema de conexión de los botones e interruptores.

3.1.6. Conector VHDCI

Una característica especial de esta placa de desarrollo es su conector VHDCI ('very-high-density cable interconnect'), en el cual se conecta la cámara utilizada en este trabajo.

Este conector de alto rendimiento permite frecuencias de varios cientos de megahercios en cada pin, gracias a sus contactos chapados en oro y por un apantallamiento completo frente a radiaciones electromagnéticas (EMI).

Para conseguir estas altas frecuencias, la PCB está ruteada de tal forma que 40 pines son 20 pares de impedancia controlada, por lo que se puede utilizar este conector como entrada de señales diferenciales de alta frecuencia, como LVDS. Los pines 15 y 49 están conectados a las señales de entrada de reloj de la FPGA.

Para la alimentación del periférico que se conecte, quedan disponibles los dos pines centrales inferiores con unas tensiones de 3,3V y 2,5V, conmutables por un 'jumper', además de los dos pines centrales superiores, que proporcionan una tensión de 5V no regulada (figura 22 y 23).

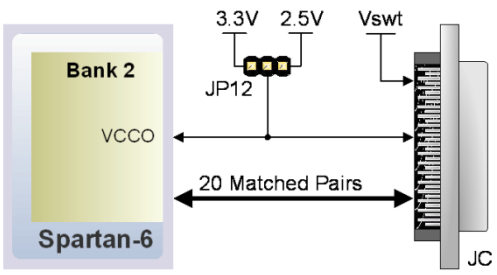

Figura 22. Esquema de conexión del puerto VHDCI.

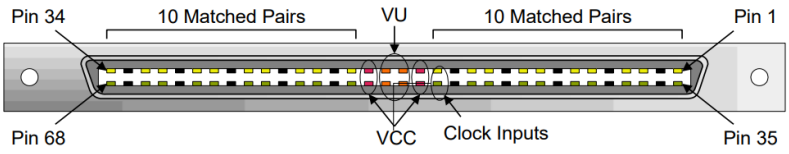

Figura 23. Detalle y disposición de pines del conector VHDCI.

3.2. Cámara

La cámara seleccionada para este trabajo ha sido un módulo desarrollado por Digilent llamado VmodCAM (Digilent, 2011).

Como se muestra en la figura 24, este módulo lleva integrados dos sensores digitales de imagen CMOS Aptina MT9D112 de dos megapíxeles (Aptina, 2005).

Los dos sensores están separados 63 mm uno de otro, para proveer una imagen estereoscópica; la posibilidad de generar una imagen tridimensional resulta muy interesante, pero no ha sido objeto de este trabajo, aunque cabe destacar que para comprender el funcionamiento del software 'demo' proporcionado por Digilent se ha experimentado con esta posibilidad.

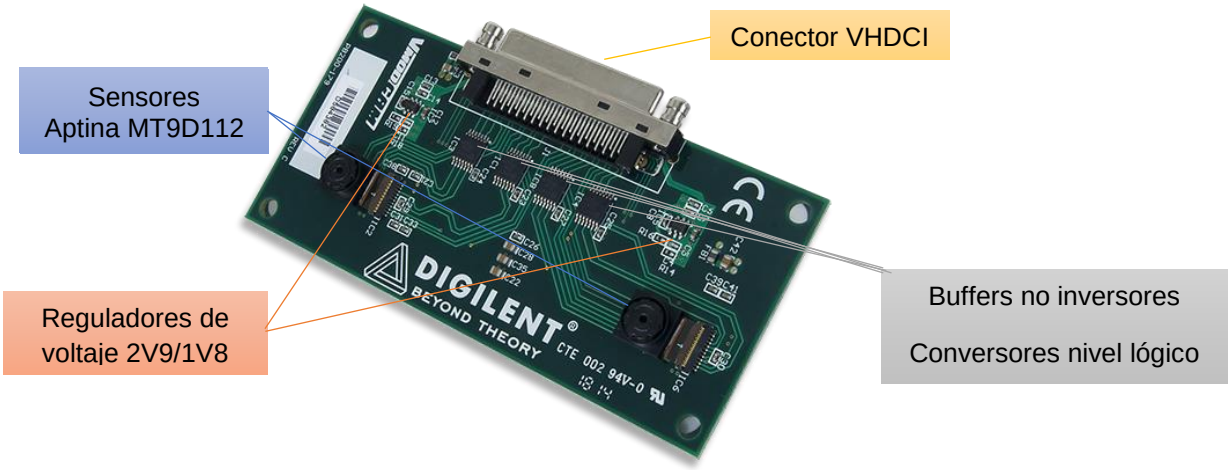

Figura 24. Módulo Digilent VmodCAM.

Características de los sensores:

- Resolución máxima de 1600x1200 píxeles.
- Profundidad de color de 10 bit en bruto.
- Control por interfaz I2C.
- Formatos de imagen:
  - ITU-R BT.601 (YCrCb)
  - RGB: 565RGB, 555RGB, 444RGB
  - Bayer
  - RAW: RAW8-bit y RAW10-bit
- Exposición automática, ganancia y balanceo de blancos.
- Escalado de imagen.
- Interfaz paralela o MIPI ('Mobile Industry Processor Interface')

- Microcontrolador, microprocesador y memorias integradas: en la figura 25 se representa cómo es todo el sistema.

Cómo se ha comentado en el punto 3.1.6, el conector utilizado por esta tarjeta es el VHDCI, muy necesario para la transmisión de alta velocidad de placa a placa.

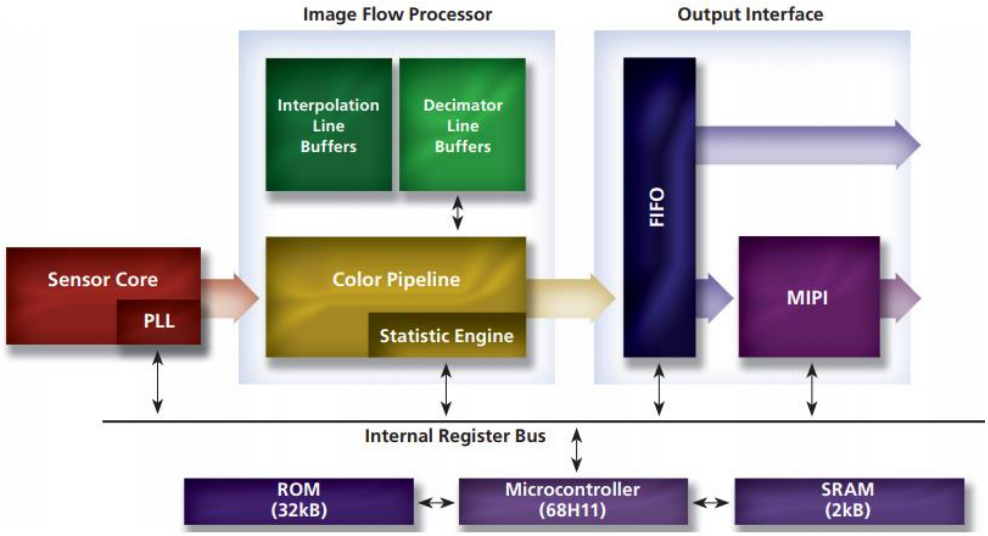

Figura 25. Diagrama de bloques del sensor MT9D112.

3.3. Diseño demo

Digilent, el fabricante tanto de la placa Atlys como la cámara VmodCAM, proporciona dos proyectos de ejemplo para el entorno Xilinx ISE específico para esta placa y cámara.

- **Proyecto 'VmodCAM_Ref_HD'**: configura las dos cámaras a una resolución de 1600x900 píxeles y la misma resolución para la señal de vídeo HDMI, permitiéndonos conmutar entre ellas utilizando el interruptor 7.
- **Proyecto 'VmodCAM_Ref_VGA_Split'**: configura ambas cámaras a una resolución de 800x600 píxeles, pero el propio sensor reescala la imagen a 640x480 píxeles.
  La señal de HDMI ofrece una resolución de 640x480 píxeles.
  Permite los tres tipos de visualización, cámara A, cámara B y ambas cámaras, siendo la mitad del fotograma izquierdo la cámara A y la otra mitad la cámara B. Estos modos se conmutan a través de diferentes combinaciones de los interruptores 6 y 7.

Este trabajo se ha realizado modificando el proyecto demo 'VmodCAM_Ref_VGA_Split', ya que en un primer momento se determinó que era menor riesgo para un procesado en directo utilizar una menor resolución. Posteriormente, a partir de otro Trabajo Fin de Grado, realizado por M. M. G. Zamora, «Estudio de un sistema de visión estéreo controlado por hardware» (Zamora, 2012), que facilitó la comprensión del sistema original, se localizaron los parámetros necesarios para cambiar la resolución deseada, por lo que la elección de un proyecto base resultó indiferente.

3.3.1. Descripción general de funcionamiento

En el momento del encendido de la FPGA se genera un reset interno en el bloque SysCon, eliminando cualquier tipo de rebotes o comportamientos inesperados. A partir de aquí, cuando el reset interno pasa a nivel lógico bajo, el resto de bloques entran en modo de funcionamiento normal.

En este mismo bloque se generan todas las señales de reloj necesarias para el sistema, así como la selección de la resolución deseada y la sincronización de los interruptores con el reloj.

De forma paralela comienzan los procesos de generación de la señal HDMI, gestión de la memoria DDR y control de la cámara. Se puede decir que los siguientes bloques en entrar en acción son los de control de las cámaras.

Estos bloques comienzan generando la temporización necesaria para el encendido de las cámaras, e inmediatamente comienza la transmisión por I2C de la configuración de las cámaras en base a parámetros almacenados en la FPGA. En el instante que termina esta transmisión las cámaras comienzan a transmitir un flujo constante de datos de imagen.

Para entender mejor cómo es el sistema, se representa mediante un diagrama en la figura 26.

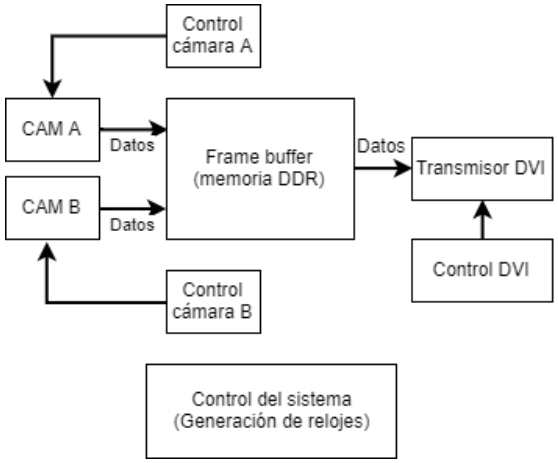

Figura 26. Esquema general del sistema original.

3.3.2. Relojes del sistema

Partiendo del oscilador externo de 100 MHz que lleva la placa Atlys se generan 11 relojes de diferentes frecuencias, pudiéndolos diferenciar en tres grupos:

- **Reloj primario:**
  - PCLK (50 MHz)

  El denominado '*primary clock',* utilizado como reloj de propósito general por la gran mayoría de procesos dentro de los bloques.

- **Relojes de control para las cámaras:**
  - CAM_CLK/CAM_CLK_180 (24 MHz)

  Estas dos señales van directamente conectadas a un bloque ODDR2 que redirecciona el reloj a un solo pin de salida, la entrada de reloj externo de una cámara.

- **Relojes de control de video:**
  - PCLK_X2 (100 MHz)
  - PCLK_X10 (250 MHz)
  - SERDESSTROBE (50 MHz)

  Necesarios para la serialización de los datos de video para conseguir las señales diferenciales TMDS utilizadas por la interfaz HDMI.

- **Relojes de control para la memoria DDR:**
  - DDR2CLK_2X/DDR2CLK_2X_180 (666,6 MHz)

- mcb_drp_clk (41,7 MHz)
- PLL_CE_0/PLL_CE_90 (333,3 MHz)

Todos estos relojes son utilizados por el bloque 'memc3_wrapper' ('mcb2_ddr2') dentro del bloque FBCtl, encargado de la gestión de la memoria DDR.

En la figura 27 podemos ver la relación de los relojes entre sí, o cómo estos son generados a partir de un reloj "superior". Todos los relojes se generan a partir de otro, por lo que este tipo de metodología es efectiva para garantizar una buena sincronización. En la misma figura se representan con bloques en forma de flecha cómo han sido generados los relojes, ya sea utilizando un DCM o un PLL de la FPGA.

Todos los relojes se encuentran '*gateados*' a la red global de relojes para disminuir las posibles derivas de reloj y retrasos.

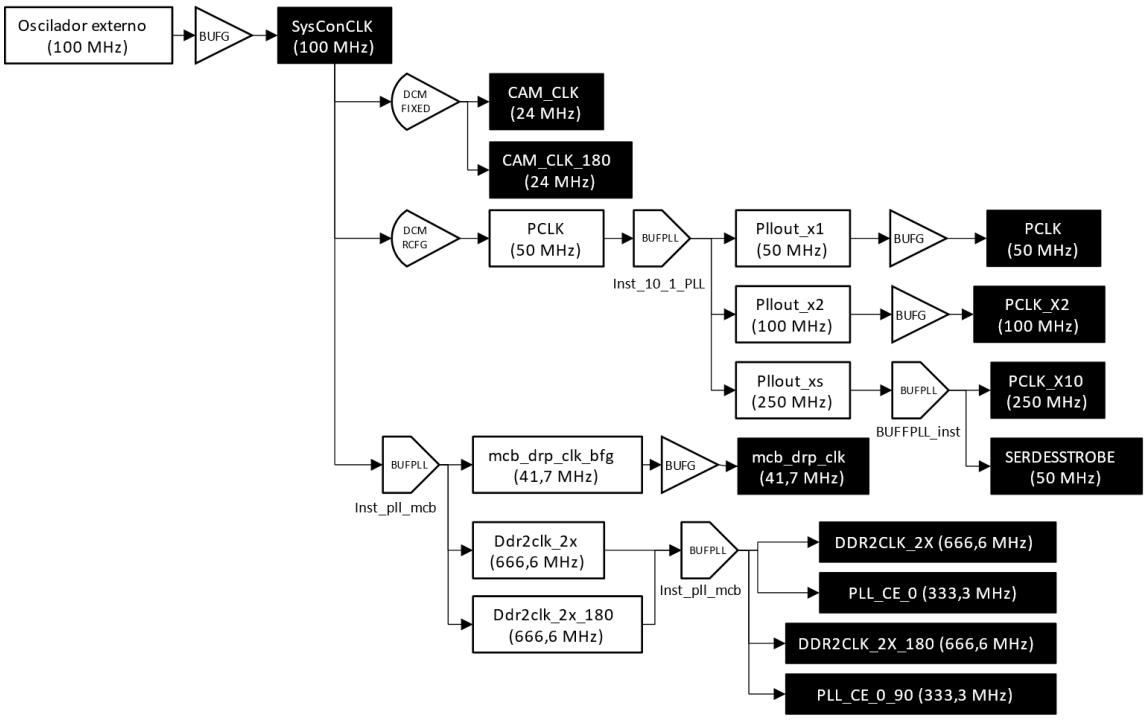

Figura 27. Árbol de relojes.

Por último, en la tabla 2 se muestran los relojes que utiliza cada bloque VHDL.

	VideoTimingCtl	CamCtl	DVITransmitter	FBCtl
CAMCLK		✓		
CAMCLK_180		✓		

DDR2CLK_2X				✓
DDR2CLK_2X_180				✓
mcb_drp_clk				✓
PCLK	✓		✓	✓
PCLK_X2			✓	
PCLK_X10			✓	
PLL_CE_0				✓
PLL_CE_90				✓
SERDESSTROBE			✓	

Tabla 2. Relación entre relojes y bloques.

3.3.3. Descripción de los principales bloques

En esta sección se va a introducir los bloques utilizados por el sistema, explicando la funcionalidad de cada uno y las relaciones entre ellos. Para mejor compresión del sistema se representa en la figura 28 un diagrama que incluye los bloques más importantes del sistema.

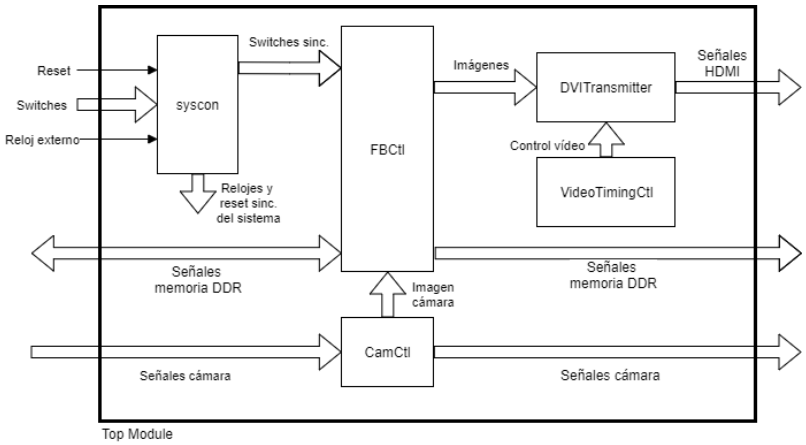

Figura 28. Diagrama de bloques del sistema simplificado.

3.3.3.1. Top Module

Bloque principal que instancia todos los demás bloques y los interconecta entre ellos. También realiza la conexión entre los pines de la FPGA y los bloques anteriores.

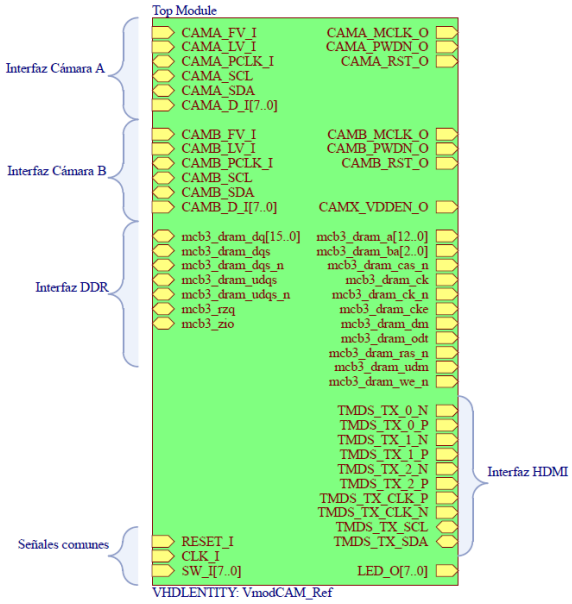

Figura 29. Esquemático del sistema completo.

3.3.3.2. SysCon

Es el encargado de generar todos los relojes del sistema, vistos en la sección 3.3.2, y generar un reset asíncrono sin rebotes de mucha fiabilidad al encenderse la FPGA y de sincronizar los interruptores con el reloj del sistema.

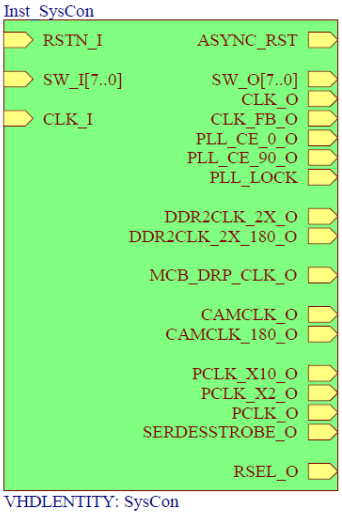

Figura 30. Esquemático del bloque SysCon.

- **dcm_fixed**
  IP core proporcionado por Xilinx. Genera una señal de reloj de 24 MHz (CAMCLK) y su opuesta en fase (CAMCLK_180) dividida de forma fija a partir del reloj principal.

- **dcm_rcfg**
  Igual que el anterior, pero permite generar una señal de reloj con una división parametrizable en función de la resolución elegida de video HDMI. Esté reloj será el reloj de entrada del PLL '*Inst_10_1_PLL'*.

- **pll_base**
  El propósito de este bloque es servir como un sintetizador de frecuencia y filtro *jitter*, tanto para relojes internos como externos.
  - **Inst_10_1_PLL:** genera los relojes necesarios para la serialización de la señal de video y el reloj primario.
    - Parámetros:
      - Periodo reloj entrada: 13,468 ns
      - Factor de multiplicación general: 10
      - Factor de división general: 1
      - Factor de división PCLK: 10

- Factor de división PCLK_X2: 10/2
- Factor de división PCLK_X10: 1

- **Inst_pll_mcb:** genera los relojes necesarios para el control de la memoria DDR
  - Parámetros:
    - Periodo reloj entrada: 13,468 ns
    - Factor de multiplicación general: 20
    - Factor de división general: 3
    - Factor de división DDR2CLK_2X: 1
    - Factor de división DDR2CLK_2X_180: 1
    - Factor de fase DDR2_CLK_2X_180: 180
    - Factor de división mcb_drp_clk: 16

3.3.3.3. VideoTimingCtl

Genera las señales de sincronización apropiadas de acuerdo con la resolución elegida de video HDMI.

Las señales VS_O y HS_O llevan a cabo la sincronización vertical y horizontal respectivamente, VDE_O es la señal de dato válido.

VCNT_O y HCNT_O contabilizan la fila y columna actual, pero no son utilizadas.

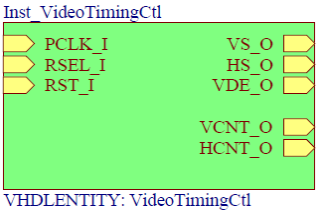

Figura 31. Esquemático del bloque VideoTimingCtl.

3.3.3.4. FBCtl

Bloque principal en la gestión de la memoria DDR. Redirecciona todas las señales de la interfaz con la memoria DDR al bloque de control de esta (‘*memc3_wrapper*’), además de comandar las FIFO de lectura y escritura en memoria, tanto de las dos cámaras (escritura) como de la imagen que se transmitirá por el puerto HDMI (lectura).

El código utiliza una constante llamada VMEM_SIZE igual a 2^{23} (bits). Toma como referencia esta constante para escribir y leer en memoria.

La cámara A es guardada en la dirección 0, al principio de la memoria, mientras que la cámara B se almacena en la dirección VMEM_SIZE.

La lectura en memoria dirigida al video por HDMI conmuta entre sumar VMEM_SIZE a la dirección actual o no, dando lugar a una conmutación entre una cámara y otra.

En la figura 33 se representa gráficamente las zonas de memoria reservadas para las imágenes.

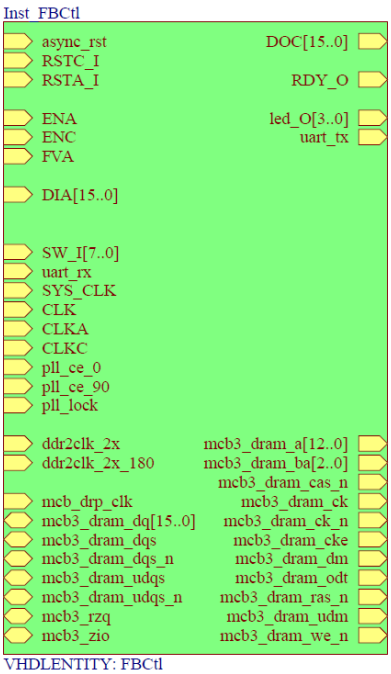

Figura 32. Esquemático del bloque FBCtl.

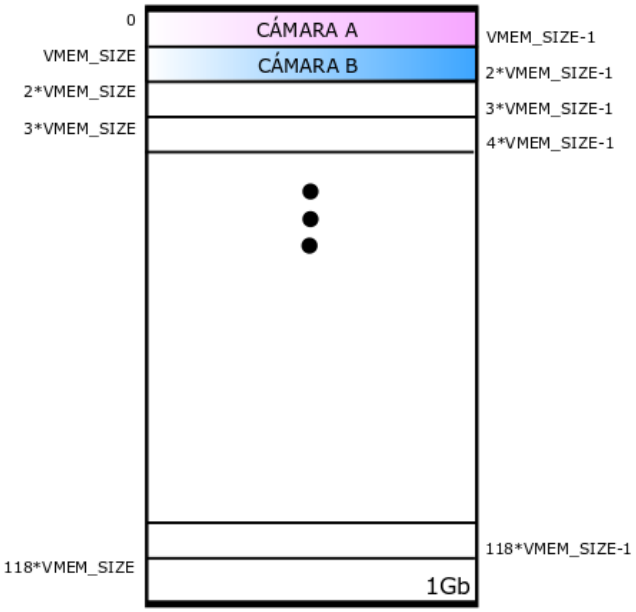

Figura 33. Mapeo de memoria del sistema original.

- **memc3_wrapper**
  Bloque generado por el MIG ('*Memory Interface Generator*') de Xilinx. En base al modelo de memoria DDR que se necesite controlar, el MIG genera un bloque acorde a las características de temporización necesarias y las entradas/salidas de la memoria. Desde el punto de vista del usuario, facilita el uso de una memoria DDR la cual se accede a través de una FIFO controlada por unos pocos comandos. En el apartado 3.5 se ampliará esta información.

DVITransmitter

Bloque encargado de convertir un vector de 16 bits, en el cual se almacena un pixel en formato RGB, al formato de señales diferenciales necesarias para transmitir video a través del puerto HDMI, en base a las especificaciones DVI 1.0.

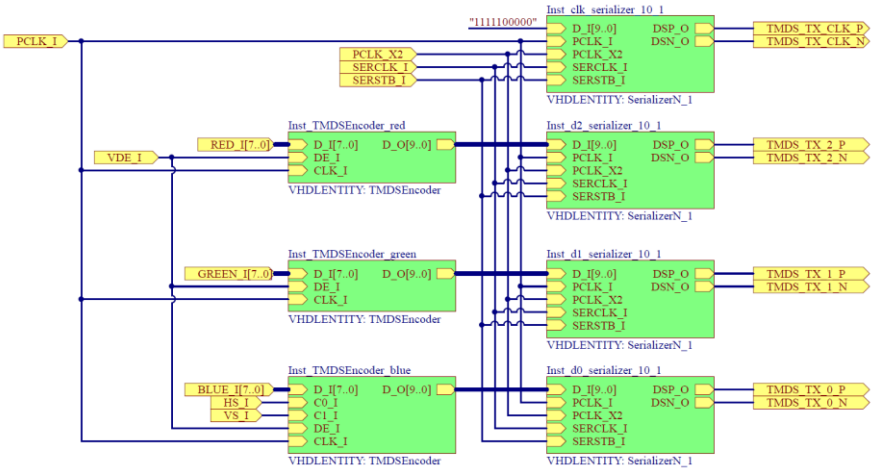

Figura 34. Esquema del bloque DVITransmitter.

- **TMDSEncoder**
  Codifica un vector de 8 bits a otro vector de 10 bits necesario por el siguiente bloque en base a las especificaciones DVI 1.0.
- **SerializerN_1**
  Bloque genérico configurado para ser un serializador 10:1. Partiendo de un vector de 10 bits, genera un par diferencial de señales que transmiten en serie.

3.3.3.5. CamCtl

Gestiona todo lo necesario para controlar las cámaras externas como: temporizaciones de reseteo inicial y encendido, transmisión por I2C de los parámetros necesarios para configurar los registros de control de las cámaras, así como implementar un *buffer* el cual almacene dos bytes consecutivos proporcionados por la cámara para disponer de un píxel completo almacenado en un vector.

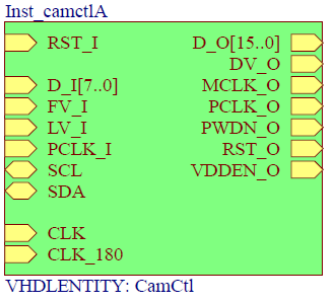

Figura 35. Esquemático del bloque CamCtl.

- **TWICtl**
  Controlador I2C el cual realiza la transmisión en este protocolo para la configuración de la cámara.

3.4. Interfaz con la cámara

Los sistemas de la cámara necesitan ser configurados correctamente primero. Esto incluye no sólo establecer los parámetros como la resolución o formato de imagen, si no también configuración del PLL y la secuenciación del microprocesador, siendo el orden en el que se realizan estos pasos muy importantes (Aptina, 2005).

3.4.1. Secuencia de encendido y reinicio.

La cámara sólo debe ser conectada a la placa una vez que las señales dirigidas por el sistema estén definidas.

La cámara usa los suministros de voltaje analógicos y digitales provistos por la placa. Los voltajes están activados por defecto, pero pueden ser apagados llevando la señal VDD-EN a nivel bajo.

Mientras que la cámara realiza el reinicio de encendido es buena idea llevar a cabo un reinicio completo como parte de la rutina de control.

Si el PLL en la cámara está activo, MCLK debe ser estable. Parando MCLK sin respetar la secuencia de reinicio puede dejar la cámara en un estado indeterminado, por lo que se recomienda realizar un ciclo de encendido en las primeras etapas del controlador.

Toda la secuencia de encendido y reinicio están representadas en las figuras 35 y 36, mientras que en la tabla 3 vemos los valores temporales necesarios para llevarlas a cabo.

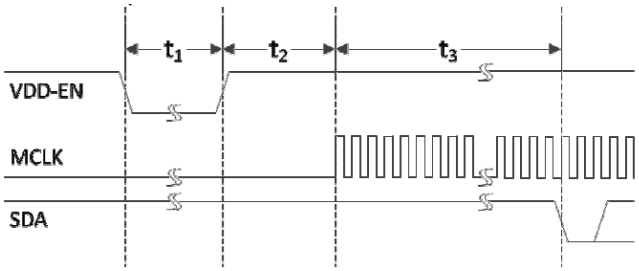

Figura 36. Secuencia de encendido.

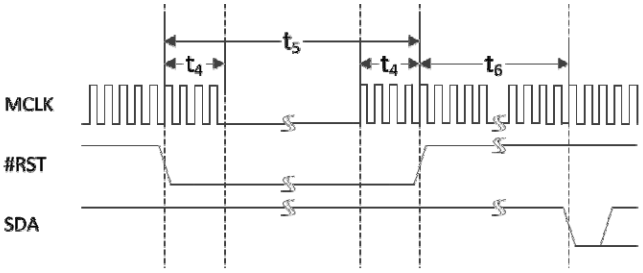

Figura 37.Secuencia de reinicio.

	Descripción	Min	Max	Unit
t1	Ancho pulso negativo VDD-EN	100		ms
t2	Separación entre nivel alto VDD-EN y primer pulso MCLK	75		us
t3 t6	Tiempo de lectura de la ROM hasta el primer byte de control		6000	Ciclos MCLK
t4	MCLK activo antes/después de flanco de #RST	10		Ciclos MCLK
t5	Pulso negativo de #RST	30		Ciclos MCLK

Tabla 3. Temporización encendido/reset.

3.4.2. Interfaz de control

Para controlar y configurar varias partes de la cámara se utiliza la interfaz I2C como vía de transferencia, actuando la cámara como esclavo.

Escritura en registro:

- Condición de inicio
- Dirección del dispositivo de 8 bit (0x78 para el sensor MT9D112) + bit de reconocimiento (ACK)

- Byte alto de la dirección del registro 16 bit + bit ACK
- Byte bajo de la dirección del registro 16 bit + bit ACK
- Byte alto del dato de 16 bit + bit ACK
- Byte bajo del dato 16 bit + bit ACK
- Condición de parada

Lectura de registro:

- Condición de inicio
- Dirección del dispositivo de 8 bit (0x78 para el sensor MT9D112) + bit ACK
- Byte alto de la dirección del registro 16 bit + bit ACK
- Byte bajo de la dirección del registro 16 bit + bit ACK
- Condición de inicio
- Dirección del dispositivo de 8 bit (0x78 para el sensor MT9D112) + bit ACK
- Byte alto del dato de 16 bit + bit ACK
- Byte bajo del dato 16 bit
- Condición de parada

Hay dos tipos de controles de configuración, registros de hardware y variables del controlador. Los registros de hardware generalmente controlan el sensor y algunos otros subsistemas, mientras que las variables secuencian el microprocesador.

Llamaremos una transferencia al envío de los 16 bit de la dirección del registro junto con los 16 bit de datos adjuntos. Generalmente, para la escritura en un registro basta con una transferencia, mientras que para una escritura en una variable se necesitan dos.

Las escrituras en variables consisten en una escritura en el registro 0x338C, la dirección de la RAM del microcontrolador, adjuntando como dato la ID del driver y la dirección de la variable a escribir. La segunda transferencia tiene como destino la dirección 0x3390, dirección del buffer de datos que se almacenarán en RAM, adjuntando como dato el valor de la variable.

Los valores de los registros y variables, de 16 bit, suelen subdividirse en grupos de varios bits o uno solo que controlan alguna función diferente.

Al realizar muchas escrituras en registros o variables es necesario una secuencia de refresco o un reinicio del sistema posteriormente.

3.4.3. Interfaz de datos

El puerto paralelo de la cámara está constituido por un bus de 8 bits de datos, una señal de reloj (PIXCLK), y las señales LINE_VALID y FRAME_VALID.

Tal y como está configurada la cámara para este proyecto, el bus transmite en cada dos ciclo de reloj un pixel a color RGB 565. Esto quiere decir que cada dos bytes enviados constan de un valor de 5 bits para el subpíxel rojo, un valor de 6 bits para el subpíxel verde y un valor de 5 bits para el subpíxel azul, tal como se muestra en la figura 38:

RGB565															
Byte alto								**Byte bajo**							
R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0

Figura 38. Empaquetado de un pixel RGB565.

Durante la transferencia de un pixel la señal LINE_VALID se mantiene en nivel alto, mientras que la señal FRAME_VALID se mantiene en nivel alto durante toda la transmisión de un fotograma. Gracias a este comportamiento, se puede utilizar la señal LINE_VALID como verificación de transferencia y la señal FRAME_VALID para sincronizar la captura con el inicio de un fotograma. La figura 38 muestra el comportamiento descrito para la transferencia de datos.

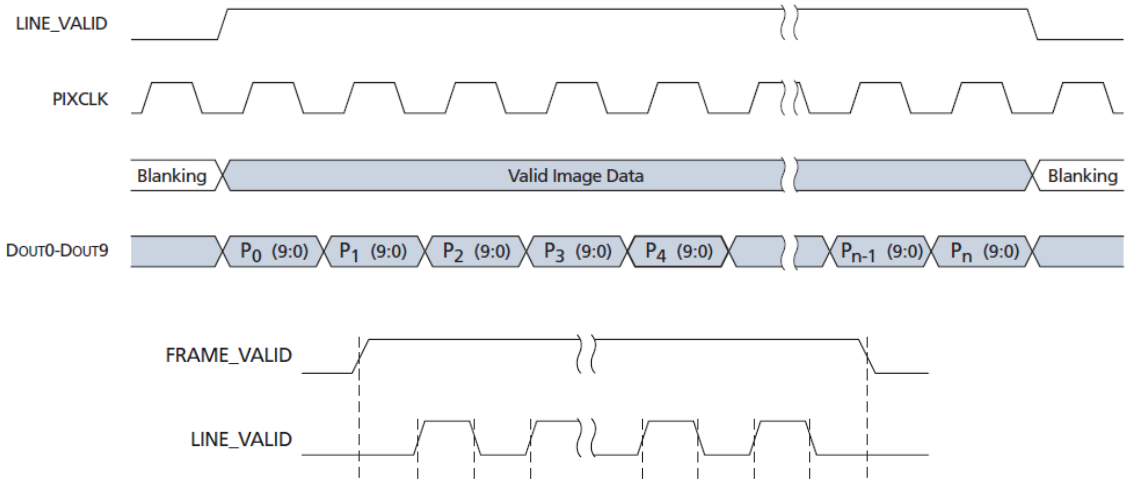

Figura 39. Temporización en el envío de una imagen.

El orden de transmisión de los pixeles es tal como se indica en la figura 40:

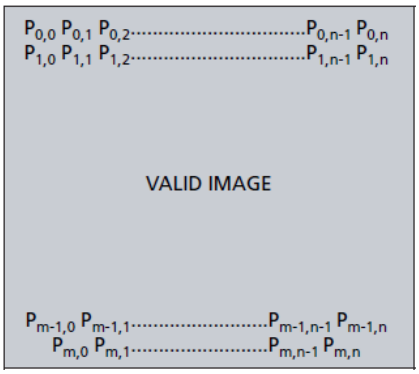

Figura 40. Orden en la transferencia de píxeles.

El primer pixel del fotograma enviado es aquel que se ubica en la esquina superior izquierda, siguiendo hacia la derecha. Una vez terminada una fila, continúa con el primer pixel de la siguiente fila.

3.5. Interfaz con la memoria DDR

DDR (‘*Double Data Rate*’) es un tipo de memoria RAM de la familia de las SDRAM (‘*Synchronous Dynamic Random-Acces Memory*’), memorias de interfaz síncrona de acceso aleatorio, siendo DDR una evolución de las SDRAM ya que permite realizar dos o más operaciones en un ciclo de reloj (en flanco de subida y en flanco de bajada).

Las memorias DDR constan de una arquitectura multibanco, en la cual cada banco del dispositivo es internamente independiente del resto. Estos bancos están formados por matrices de bits divididos en filas y columnas (Deutron Electronics Corp.).

3.5.1. Señales

A continuación, se enumeran las señales más importantes.

- CKE: Clock Enable.
- CS: Chip Select.
- DQ: Data Mask
- RAS: Row Address Strobe.
- CAS: Column Address Strobe.
- WE: Write Enable.
- RST: Reset.

- BAn: Bank Adress. Depende del número de bancos de la memoria.
- An: Addressing. Depende de la configuración de la memoria.
- DQ: Bus de datos bidireccional. Depende de la anchura de datos de la memoria.

3.5.2. Controlador

El control de una memoria DDR resulta bastante complejo y poco amigable de cara al usuario, precisa de muy buena temporización de las señales respecto al reloj, lidiar con señales bidireccionales y evitar conflictos en las lecturas y escrituras. Por ello son comúnmente utilizados controladores de memoria DDR, pudiendo ser dispositivos físicos (IC) o bloques de control dentro de un dispositivo FPGA, como ocurre en este caso.

Básicamente consisten en dos grandes partes y sub-bloques:

- **Interfaz principal**: proporciona interfaz con el resto del sistema, siendo independiente del tipo de memoria utilizada. Procesa comandos, recibe los datos a escribir y devuelve los datos de la lectura.
  - **Mapeo de memoria**: decodifica una dirección de memoria en banco, fila y columna.
  - **Arbitrador**: elige el orden en el cual se accede a memoria.
- **Interfaz trasera**: proporciona la interfaz específica con la memoria utilizada.
  - **Generador de comandos**: activa las señales específicas para una memoria en concreto con su respectiva temporización en función del comando a realizar.

3.5.3. Bloque controlador FPGA

Las FPGA de la familia Spartan-6 pueden contener hasta cuatro MCB (‘*Memory Controller Block*’) (Xilinx, 2010) (Xilinx, 2012). Los MCB son bloques empotrados multi-puerto dedicados al control de memoria que simplifican la interfaz con los dispositivos de memoria estándares más populares. Podemos ver el sistema de un bloque controlador en la figura 40.

Características más importantes:

- Soporte de memorias estándar DDR, DDR2, DDR3 y LPDDR.
- Hasta un rendimiento de 800 Mb/s.
- Interfaces de datos de 4-bit, 8-bit o 16-bit.
- Memorias de hasta 4 Gb.

- Interfaz de usuario multi-puerto configurable.
  - De 1 a 6 puertos por MCB dependiendo de la configuración.
  - Opciones de bus de datos de 32, 64 o 128 bit.
  - Puertos bidireccionales o unidireccionales (lectura o escritura)

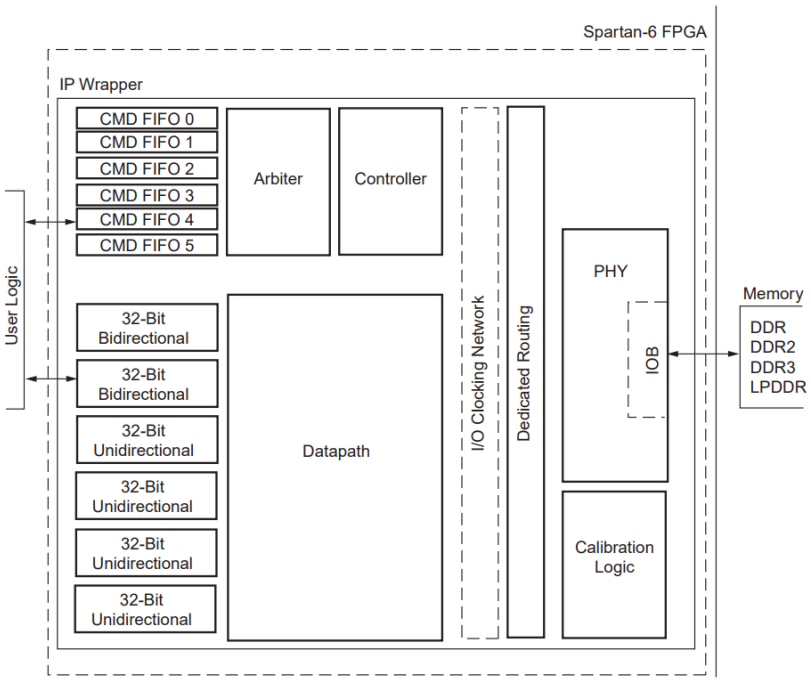

Figura 41. Diagrama del controlador de memoria DDR.

Para crear el bloque controlador se utiliza la herramienta MIG (‘*Memory Interface Generator*’) incluida en el *Core Generator* de Xilinx. Esta herramienta nos proporciona la facilidad de configurar el bloque de control de memoria a través de una interfaz gráfica. También nos proporciona la configuración de una serie de dispositivos de memoria que Xilinx dispone en una base de datos.

3.5.3.1. Interfaz de usuario

Hay tres tipos de puertos básicos disponibles para el usuario:

- Puertos de lectura (unidireccional)
- Puertos de escritura (unidireccional)
- Puertos de lectura y escritura (bidireccional)

Cada puerto contiene un camino para comandos y otro para datos. Para los puertos unidireccionales, el camino de comandos está emparejado con el camino de datos de solo lectura o escritura, sin embargo, para los puertos bidireccionales el camino de comandos está compartido con datos de escritura o lectura.

Los puertos disponen de FIFOs que almacenan información de las instrucciones de los comandos realizados (dirección, lectura o escritura, datos, etc.) y ejercen de puente entre el dominio temporal de la interfaz de usuario y el del controlador de la memoria.

Señales más importantes disponibles por el usuario:

- **Comandos**
  - pX_cmd_addr: Dirección de inicio de la tranferencia.
  - pX_cmd_bl: Longitud de la ráfaga de datos en palabras de usuario.
  - pX_cmd_clk: Reloj para el FIFO de comandos
  - pX_cmd_empty: bandera que indica que el FIFO de comandos no contiene comandos en cola.
  - pX_cmd_en: señal de habilitación del comando.
- **Escritura**
  - pX_wr_clk: Reloj utilizado por el FIFO de datos de escritura.
  - pX_wr_data: valor del dato a almacenar en el FIFO de datos de escritura.
  - pX_wr_en: señal de habilitación que indica un dato válido en pX_wr_data.
- **Lectura**
  - pX_rd_clk: Reloj utilizado por el FIFO de datos de lectura.
  - pX_rd_data: valor del dato a almacenar en el FIFO de datos de lectura.
  - pX_rd_en: señal de habilitación que indica un dato válido en pX_rd_data.

3.6. Interfaz HDMI

Aunque se podría tratar como una interfaz HDMI basada en su protocolo específico, en realidad se trata de una interfaz DVI sobre un conector HDMI ya que así ha sido programada la FPGA.

Estas dos interfaces son compatibles entre sí, se puede decir que HDMI es una ampliación de la especificación DVI 1.0, limitándose a una codificación RGB para los píxeles y sin datos más allá de imagen, de ahí que una conexión DVI no pueda transmitir audio y una HDMI sí.

El formato de datos estándar utilizado es RGB 24 bits MSB, 8 bits por color.

3.6.1. TMDS

La interfaz DVI utiliza señales diferenciales de transición minimizada (‘*Transition Minimized Differential Signaling*’) como base para la interconexión eléctrica.

La transición minimizada se consigue implementando un algoritmo de codificación que convierte 8 bit de datos en 10 bit, llamada codificación 8b/10b. Con esta codificación se consigue una transmisión compensada en continua (‘*DC balanced’)*, gracias a que la diferencia entre el número de unos y ceros en una cadena de al menos 20 bits es no más de dos y de que no se transmiten más de cinco unos o ceros consecutivos (Digital Display Working Group, 1999).

3.6.1.1. Arquitectura

Un transmisor TMDS codifica, serializa y transmite un flujo de datos de entrada de 24 bits a través del enlace, pudiendo ser tanto píxeles como datos de control dependiendo del estado de la señal ‘data enable’ (DE). Un estado alto de esta señal indica al codificador que se pretende enviar un píxel, ignorando cualquier señal de control.

El transmisor contiene tres codificadores idénticos, uno por cada canal TMDS (para los subpíxeles rojo, verde y azul).

Cada codificador produce un flujo de datos TMDS en base a la entrada 8 bits de datos, dos entradas de control y una señal de ‘data enable’.

La mayor parte del tiempo se envían datos de imagen a través del enlace, pero por especificación, una vez cada 20 ms cómo máximo se deberá enviar codificadas la sincronización vertical (VSYNC) y horizontal (HSYNC). En la figura 41 se representa el diagrama de bloques de un enlace TMDS monocanal.

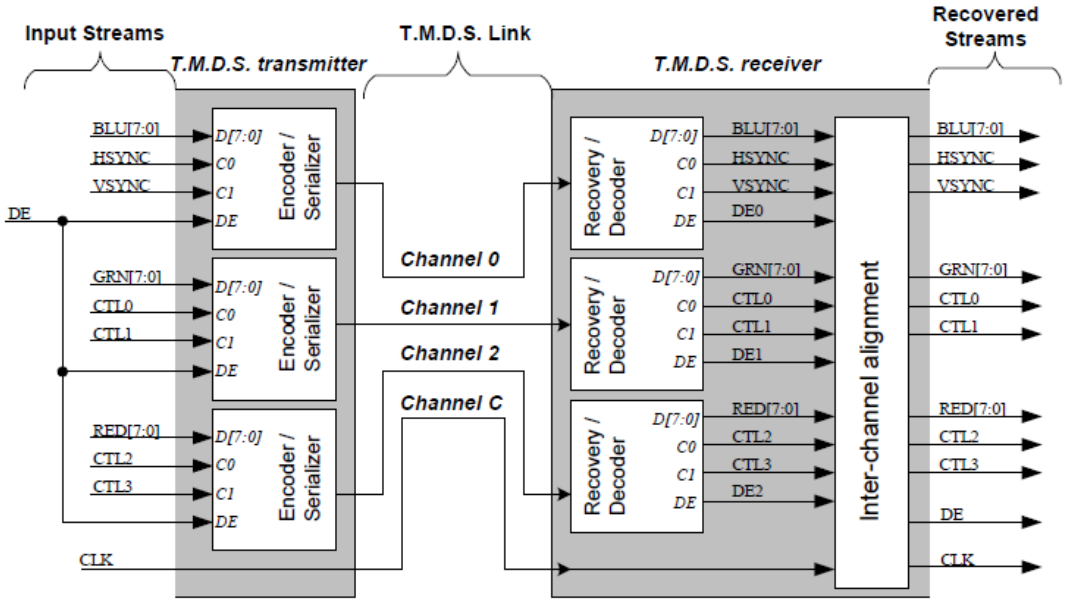

Figura 42. Diagrama de un enlace TMDS

3.6.1.2. Codificación

Cada canal TMDS de datos está dirigido por un flujo continuo de caracteres de 10 bit TMDS. La codificación es necesaria para proveer una señal balanceada por la reducción en las transiciones de la señal.

El proceso de codificación puede verse como dos etapas. La primera etapa produce una palabra de 9 bit con transiciones minimizadas a partir de la entrada de 8 bit. La segunda etapa produce una palabra de 10 bit balanceada en continua de toda la transmisión del flujo de caracteres TMDS.

La palabra de 9 bit producida por la primera etapa está formada por la representación de 8 bit de las transiciones encontradas por los 8 bit de entrada más una bandera que indica cuál de los dos métodos ha sido usado para describir las transiciones. El método para producir esta palabra es realizar operaciones XOR o XNOR de cada bit de entrada.

La segunda etapa invierte selectivamente los 8 bit de datos de la palabra de 9 bits producida por la primera etapa. Si se han trasmitido demasiados ‘unos’ y la entrada contiene más unos que ceros, la palabra codificada es invertida.

En la figura 43 se muestra todo el algoritmo de codificación TMDS, detallado secuencialmente.

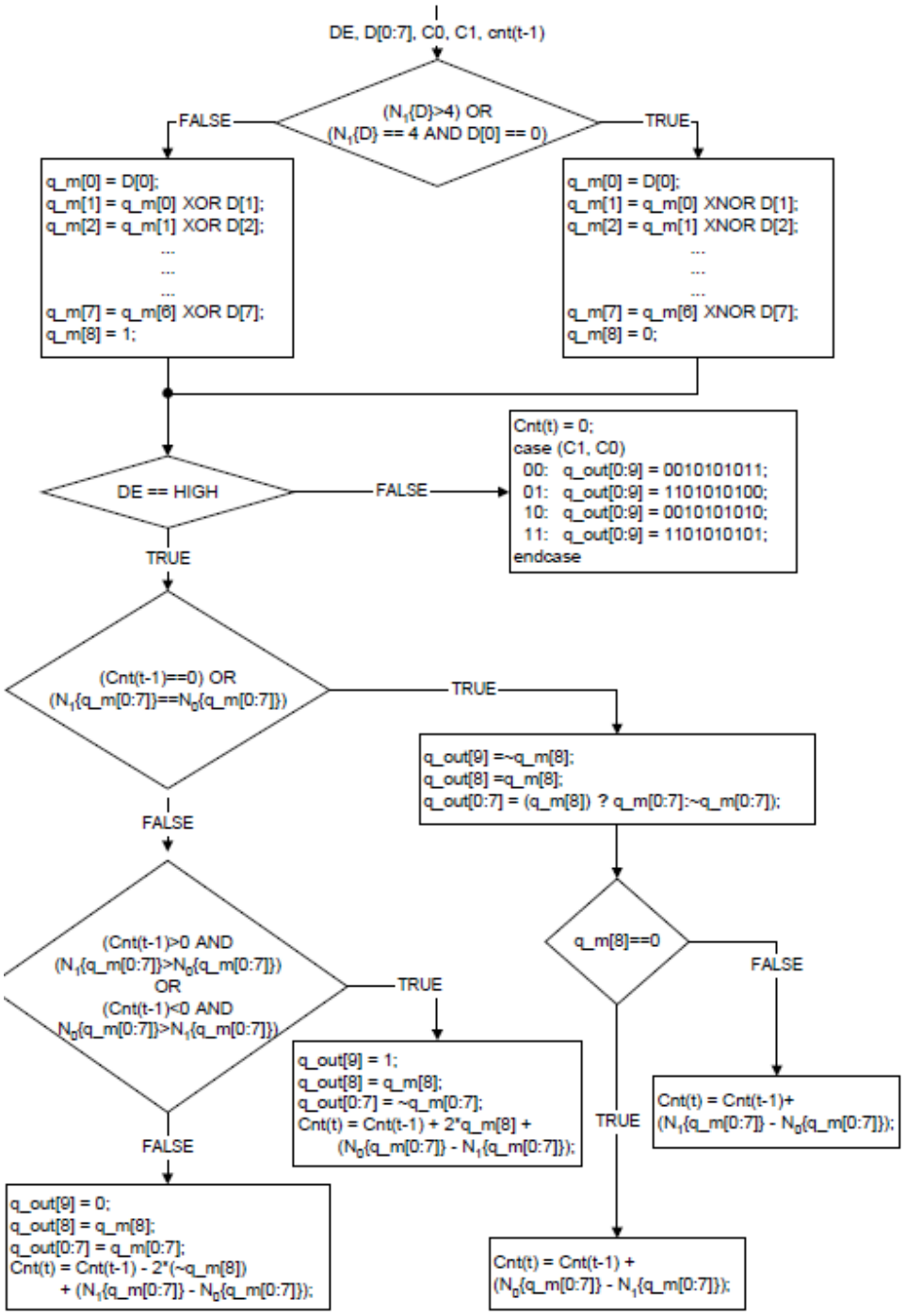

Figura 43. Algoritmo de codificación TMDS.

4. PROCESADO PARA DETECCIÓN DE BORDES

En este capítulo se detalla el sistema de filtrado de imágenes elegido, así como todas las modificaciones necesarias del sistema sobre el que se basa el proyecto.

Para finalizar se explica que método de comprobación de los resultados ha sido utilizado, necesario para la verificación de que todo el sistema funciona correctamente.

4.1. Introducción teórica

Como hemos visto en el apartado 2.2.3.2, sección “Técnicas basadas en gradiente. Método de detección de bordes”, el método para la detección de bordes de una imagen utilizando el operador Sobel, consiste en realizar la convolución de todos los pixeles de una ventana de una imagen frente a dos máscaras, máscara gradiente horizontal (H_x) y máscara gradiente vertical (H_y). Finalmente se suman los valores absolutos de los resultados de ambas convoluciones, dando como resultado el pixel de la imagen filtrada. Esta operación se repite para todos los pixeles de la imagen.

La convolución de una ventana 3x3 de una imagen bidimensional traducida a operaciones básicas matemáticas no es más que el sumatorio de las multiplicaciones de cada pixel con el respectivo valor de la máscara que le corresponde por posición:

$$g = A \otimes H = \sum_{i=-1}^{1} \sum_{j-1}^{1} A_{ij} \times H_{ij}$$

$$g = \begin{bmatrix} A_{(-1,-1)} & A_{(0,-1)} & A_{(1,-1)} \\ A_{(-1,0)} & A_{(0,0)} & A_{(1,0)} \\ A_{(-1,1)} & A_{(0,1)} & A_{(1,1)} \end{bmatrix} \times \begin{bmatrix} H_{(-1,-1)} & H_{(0,-1)} & H_{(1,-1)} \\ H_{(-1,0)} & H_{(0,0)} & H_{(1,0)} \\ H_{(-1,1)} & H_{(0,1)} & H_{(1,1)} \end{bmatrix}$$

$$= \left(A_{(-1,-1)} \times H_{(-1,-1)}\right) + \left(A_{(0,-1)} \times H_{(0,-1)}\right) + \left(A_{(1,-1)} \times H_{(1,-1)}\right)$$
$$+ \left(A_{(-1,0)} \times H_{(-1,0)}\right) + \left(A_{(0,0)} \times H_{(0,0)}\right) + \left(A_{(1,0)} \times H_{(1,0)}\right)$$
$$+ \left(A_{(-1,1)} \times H_{(-1,1)}\right) + \left(A_{(0,1)} \times H_{(0,1)}\right) + \left(A_{(1,1)} \times H_{(1,1)}\right)$$

donde A es la ventana, H la máscara y g el pixel resultante.

Cómo se puede observar en la figura 43, los pixeles de los extremos de la imagen no pueden ser procesados. Es posible añadir dos filas y columnas adyacentes de ceros, pero en este proyecto por sencillez se han obviado los bordes.

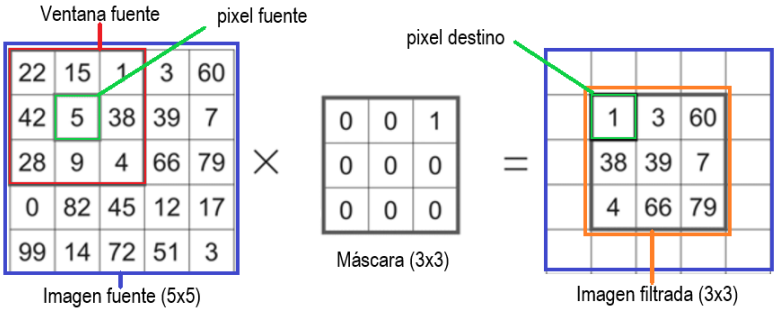

Figura 44. Aplicación de máscara de convolución.

El operador de Sobel define dos máscaras, una por cada dirección x e y, las cuales son:

$$H_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad H_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Finalmente, conociendo cómo realizar una convolución de una ventana de la imagen y el operador Sobel, las ecuaciones para obtener un pixel filtrado son:

$$g_x = A \otimes \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad g_y = A \otimes \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$g = |g_x| + |g_y|$$

4.2. Modificaciones sobre el sistema original

La resolución de todo el proyecto, tanto la imagen de salida de la cámara como la del video HDMI se mantiene en **640x480** (VGA).

El software demo de Digilent utiliza las dos cámaras que contiene la placa VmodCAM por lo que se eliminó todas las referencias a la cámara B. Esto implica borrar todas las entradas y salidas de esta cámara y su bloque de control Inst_camctlB.

La mayor parte de los cambios necesarios para el proyecto se ha llevado a cabo en el bloque FBCtl, que es el encargado de la lectura y escritura en memoria DDR de las imágenes capturadas por las cámaras.

En la figura 45 se muestra cómo queda el sistema tras realizar los cambios necesarios.

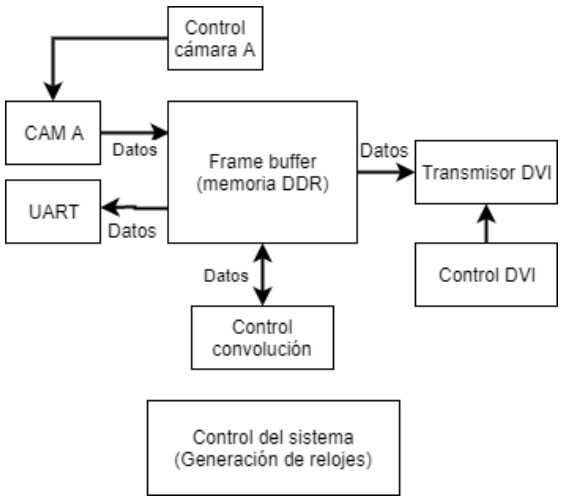

Figura 45. Diagrama de bloques del sistema modificado.

4.2.1. Modificaciones generales

El primer paso consistió en convertir un pixel RGB565 a escala de grises 8 bit. Este proceso lo podría haber llevado a cabo el procesador de la cámara, pero se prefirió que lo realizara la FPGA para poder evaluar las posibilidades de procesado en tiempo real. De forma combinacional realiza la siguiente operación:

```
redPixel <= conv_integer (DIA(15 downto 11));
greenPixel <= conv_integer (DIA(10 downto 5));
bluePixel <= conv_integer (DIA(4 downto 0));

grayPixel <= conv_std_logic_vector( ((redPixel*8*30/100) +
                                     (greenPixel*4*59/100) +
                                     (bluePixel*8*11/100)) ,8);
```

En segundo lugar, dado que el software demo utiliza datos de 32 bit como interfaz con el FIFO de escritura con la memoria DDR y ahora disponemos de pixeles en escala de gris de 8 bits, se multiplexan 4 pixeles correlativos en una palabra de 32 bit tal como se puede ver en la figura 46.

32 bits			
8 bits (MSB)	8 bits	8 bits	8 bits (LSB)
p_{n+3}	p_{n+2}	p_{n+1}	p_n

Figura 46. Empaquetado de 4 píxeles en palabra de 32 bits.

```
------------------------------------------------------------
--
-- 8-bit selection mux
------------------------------------------------------------
--
    WROUTSEL_PROC_A: process (CLKA)
   begin
      if Rising_Edge(CLKA) then
         if (SRstA = '1') then
            pa_wr_data_sel <= "00";
         elsif (ENA = '1') then
                pa_wr_data_sel <= pa_wr_data_sel + 1;
         end if;

            if (ENA = '1') then
                if pa_wr_data_sel = "00" then
                    p2_wr_data(7 downto 0) <= grayPixel;
                elsif pa_wr_data_sel = "01" then
                    p2_wr_data(15 downto 8) <= grayPixel;
                elsif pa_wr_data_sel = "10" then
                    p2_wr_data(23 downto 16) <= grayPixel;
                end if;
            end if;
      end if;
   end process;
p2_wr_data(31 downto 24) <= grayPixel;
```

4.2.2. Mapeo de memoria

En el sistema existen cinco espacios reservados de memoria del mismo tamaño. En los primeros dos espacios se almacenan de forma alterna la imagen procedente de la cámara. La razón de alternar esta dirección de memoria se debe a evitar que se sobreescriba la imagen por el siguiente fotograma mientras se está accediendo a esa zona de memoria durante el proceso de filtrado por el bloque *convolution_control*.

El tercer espacio está reservado para la imagen resultante de la convolución.

Por último, en los espacios cuatro y cinco se almacena de forma temporal un fotograma de la imagen sin filtrar y de la imagen filtrada para posteriormente enviar por UART.

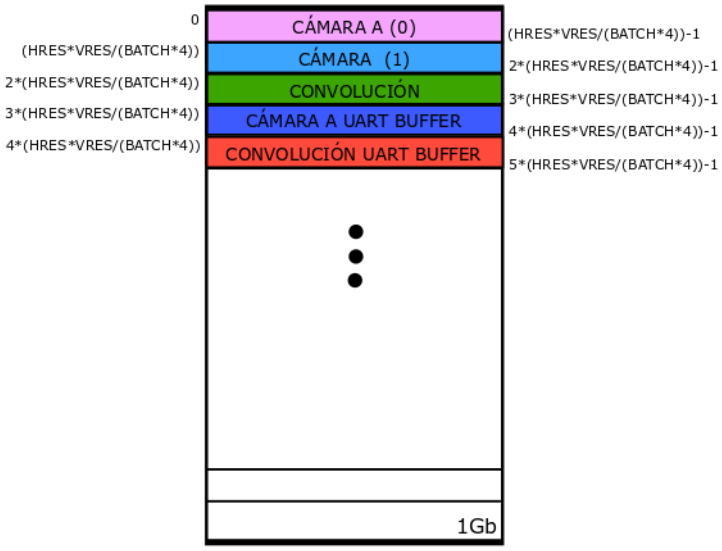

Figura 47. Mapeo de memoria del sistema.

Para todo el sistema se realizan accesos de lectura y escritura por lotes de 64 píxeles, guardados en palabras de 32 bit, por lo que el direccionamiento de memoria siempre se hace con referencia a (HRES*VRES)/(BATCH*4). Por ejemplo, si se desea acceder a los segundos 64 píxeles de la imagen filtrada la dirección sería 2*(HRES*VRES)/(BATCH*4)+2.

El valor de BATCH de la imagen anterior es de 16. El valor de HRES es 640 y el de VRES de 480.

4.2.3. Control de la convolución

Este bloque (figura 48) realiza la convolución completa de una imagen apoyado por los bloques ‘convolution’, ‘simpleDDRread’ y ‘simpleDDRwrite’. Controla todo el proceso de la convolución coordinando estos bloques.

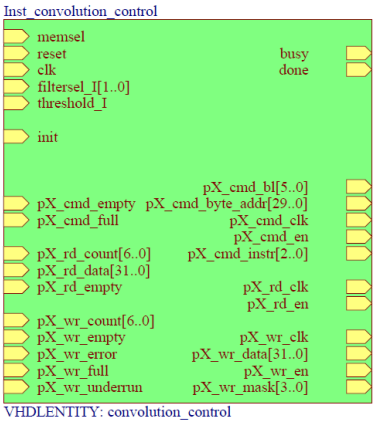

Figura 48. Esquemático del bloque convolution_control.

Puede estar en cuatro estados: inactivo, lectura, convolución y escritura.

Durante la lectura realiza accesos a memoria DDR, cargando en la RAM de la FPGA tres lotes de píxeles, correspondiente a la fila actual, la anterior y la siguiente. Los lotes de lectura cuentan con 65 píxeles debido a que para realizar la última convolución del lote es necesario el primer píxel del siguiente lote, por lo que se lee un píxel más. Análogamente, se almacena temporalmente el último pixel del lote anterior para realizar la convolución del primer píxel del siguiente lote.

Posteriormente realiza la convolución del lote de píxeles almacenando en RAM el pixel resultante de cada operación. Una vez procesado todo un lote se escribe en DDR el lote de píxeles procesados.

Durante el procesado de una fila existen tres casos que es necesario diferenciar. El más común es realizar la convolución de un píxel al que le rodean píxeles dentro del mismo lote. Otro caso es realizar la convolución de del último píxel de un lote, en este caso se utiliza ese píxel adicional que se obtuvo en la lectura. Por último, la convolución del primer píxel de un lote diferente al primero de una fila, que utiliza los últimos pixeles del anterior lote sin volverlos a leer de memoria DDR. En la figura 49 se presentan los conceptos de ‘anterior’, ‘actual’ y ‘siguiente’ para los píxeles y lotes, además de los casos antes mencionados.

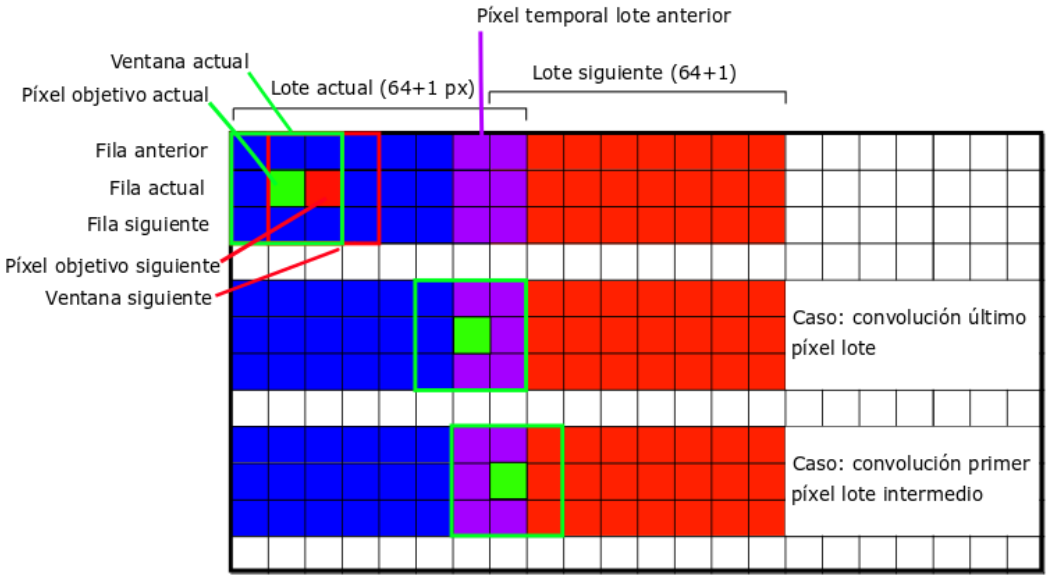

Figura 49. Detalles del proceso de filtrado de una imagen.

Los casos de los píxeles extremos no son especiales ya que los bordes no son tratados.

Todo este ciclo es repetido hasta completar una fila, después pasa a la siguiente fila, así hasta completar toda la imagen, en ese momento se activa la señal 'done' para declarar que se ha terminado el procesado. En la figura 50 se detalla todo el algoritmo en pseudocódigo.

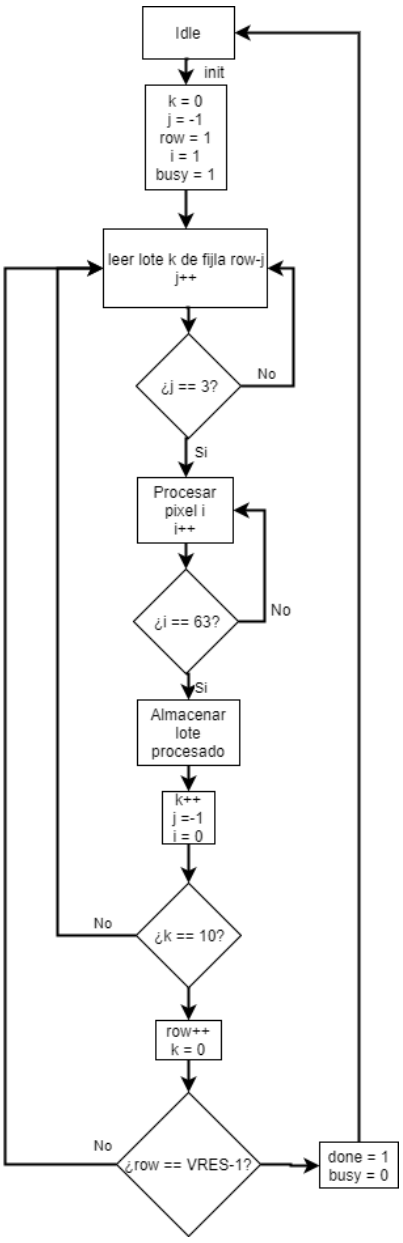

Figura 50. Algoritmo del bloque convolution_control

4.2.4. Convolución

El bloque (figura 51) “convolution” es el encargado de realizar la convolución de una ventana 3x3. Cuenta con 9 entradas de datos 8 bits, los píxeles de la ventana, y una salida 8 bits, que es el pixel resultante.

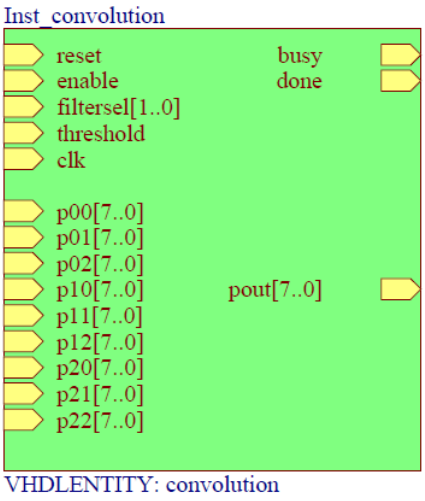

Figura 51. Esquemático del bloque convolution.

Además de las entradas y salidas convencionales de control, cuenta con la entrada ‘*filtersel’*, que selección el kernel de la convolución entre dos disponibles. Para *‘filtersel’* igual a “01” selecciona el kernel de Emboss y para cualquier otro valor el de Sobel.

Por último, la entrada ‘*threshold’* habilita un umbral en el que si el píxel resultado es menor o igual a 55 se iguala directamente a cero. Este proceso equivale a un filtro paso baja, eliminando ruido de alta frecuencia y resultando una imagen más limpia sin falsos bordes o granulado en zonas planas.

Todo el proceso se realiza en 6 pasos secuenciales (figura 52). En un primer momento se intentó que la FPGA realizara todas las sumas y multiplicaciones en un solo ciclo de reloj, pero no resultó viable ya que se superaban los DSP Slices disponibles. Se decidió optar por una secuencia de 4 etapas en las que se realizan simultáneamente 6 multiplicaciones, una multiplicación por fila y kernel (x e y), y suma del valor anterior obtenido.

Después de realizar las multiplicaciones se suman los resultados de cada fila y por último se suman los absolutos de las multiplicaciones de ambos kernels, evitando saturación del pixel resultante. Complementariamente se aplica el umbral antes descrito si está habilitada esta opción.

Si pixel > 255; pixel = 255.

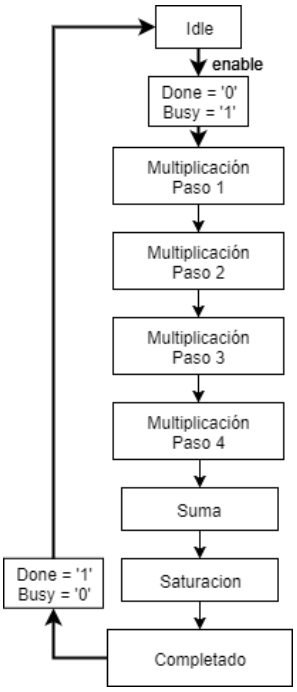

Figura 52. Diagrama del bloque convolution.

4.2.5. Lectura/escritura DDR

El propósito de estos dos bloques (figura 53) es de acomodar la interfaz del acceso a memoria DDR con el sistema. Se puede entender que ambos bloques actúan de buffer, tanto como para realizar escritura como del resultado de una lectura, y de máquina de estados para realizar la temporización y esperar las señales necesarias en el orden adecuado.

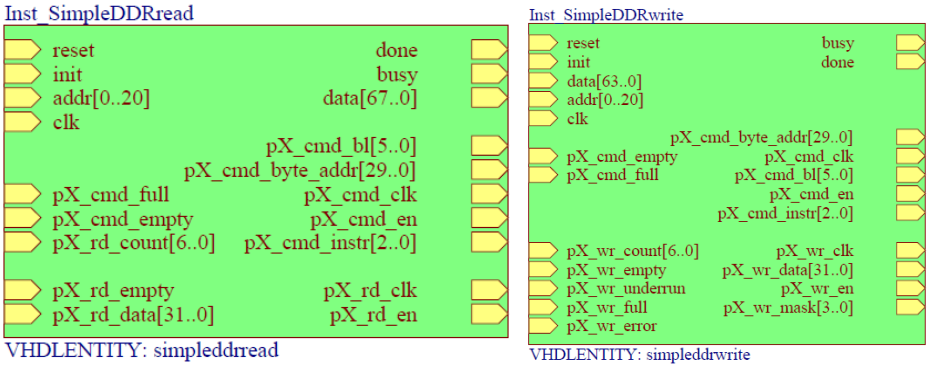

Figura 53. Esquemáticos de los bloques simpleDDRread y simpleDDRwrite.

Ambos bloques reciben y gobiernan directamente las señales correspondientes del bloque MIG generado por Xilinx para el control de memoria DDR.

Disponen de las típicas señales de control convencionales del proyecto, además de:

- SimpleDDRread:
  - addr[0..20]: Entrada. Natural. Puntero a la dirección de memoria más baja que se desea leer.
  - data[67..0]: Salida. Array con 68 bytes resultantes de 17 lotes de datos.

- SimpleDDRwrite:
  - addr[0..20]: Entrada. Natural. Puntero a la dirección de memoria más baja que se desea escribir.
  - data[63..0]: Entrada Array con 64 bytes que componen 16 lotes de datos.

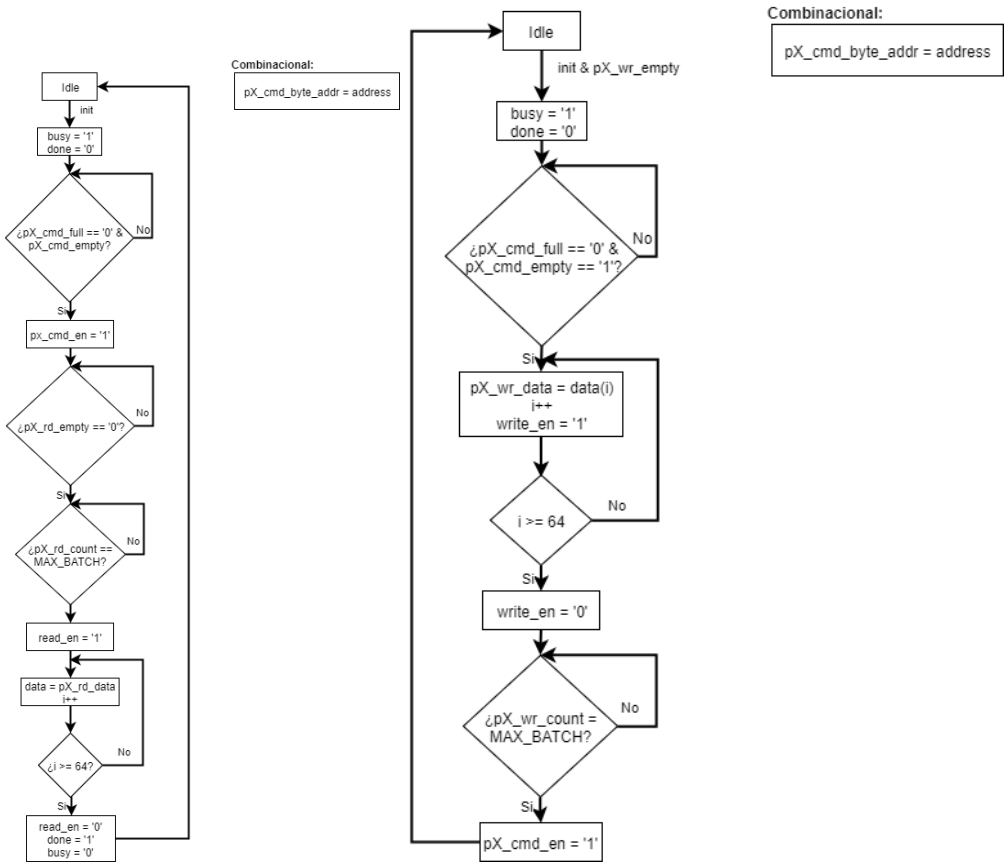

Figura 54. Algoritmos de los bloques simpleDDRread y simpleDDRwrite.

4.3. Comprobación de resultados

4.3.1. Introducción

Una vez se montó todo el sistema con éxito y se pudo visualizar en pantalla la imagen procedente de la cámara filtrada, surgió una duda: ¿era un resultado correcto?

Para ello se pensó un sistema que realizara el mismo algoritmo, pero en un entorno controlado: Matlab, por lo que se programó una aplicación con interfaz gráfica (GUI) que es la encargada de la recepción de las imágenes, réplica del procesado y comparación de los resultados.

4.3.2. Transferencia

En primer lugar, era necesario transferir tanto un fotograma de la imagen original como el resultado del procesado del mismo fotograma.

Para ello se incluyó un bloque IP que utiliza la UART incluida en la placa de desarrollo para realizar una transmisión serie hacia un PC a través del puerto USB específico para esta tarea.

Gobernando este bloque existe un bloque de control que lo sincroniza con otras instancias de los bloques '*simpleDDRread*' y '*simpleDDRwrite*'.

Este bloque de control está a la espera de la recepción de un comando especial por UART que le indique el inicio de la secuencia de envío de los fotogramas. En el momento que es detectado procede a copiar tanto el fotograma actual enviado por la cámara como el resultante del filtrado a una zona superior de memoria para poder ser enviado por UART a una velocidad relativamente lenta.

Una vez son copiados los dos fotogramas comienza la transferencia de un total de 614.400 bytes (600 KB) a una velocidad de 115200 bps.

4.3.3. Procesado y visualización

En el otro extremo, un PC con Matlab recibe las imágenes gracias a una aplicación especialmente programada en Matlab, con interfaz gráfica GUIDE, para este proyecto.

El algoritmo del proceso de detección de bordes es el mismo que el utilizado en la FPGA:

```
Kx = int8([-1 0 +1;-2 0 +2;-1 0 +1]);
Ky = Kx';

for y = 2:639
   for x = 2:479
        Im = [Img1(x-1,y-1) Img1(x-1,y) Img1(x-1,y+1);
              Img1(x,y-1) Img1(x,y) Img1(x,y+1);
              Img1(x+1,y-1) Img1(x+1,y) Img1(x+1,y+1)];

        Gx = 0;
        Gy = 0;
        for j=1:3
            for i=1:3
                Gx = Gx + int16(Im(i,j))*int16(Kx(i,j));
                Gy = Gy + int16(Im(i,j))*int16(Ky(i,j));
            end
        end
        G2 = abs(Gx) + abs(Gy);
        if threshold == 1 && G2 <= 55
           G2 = 0;
        end
        Img3(x,y) = uint8(G2);
   end
end
```

La interfaz gráfica (figura 55) se compone de esencialmente tres ventanas de visualización. En la primera se muestra la imagen sin procesar de la cámara, en la segunda la imagen procesada por la FPGA y en la tercera la imagen procesada por el PC.

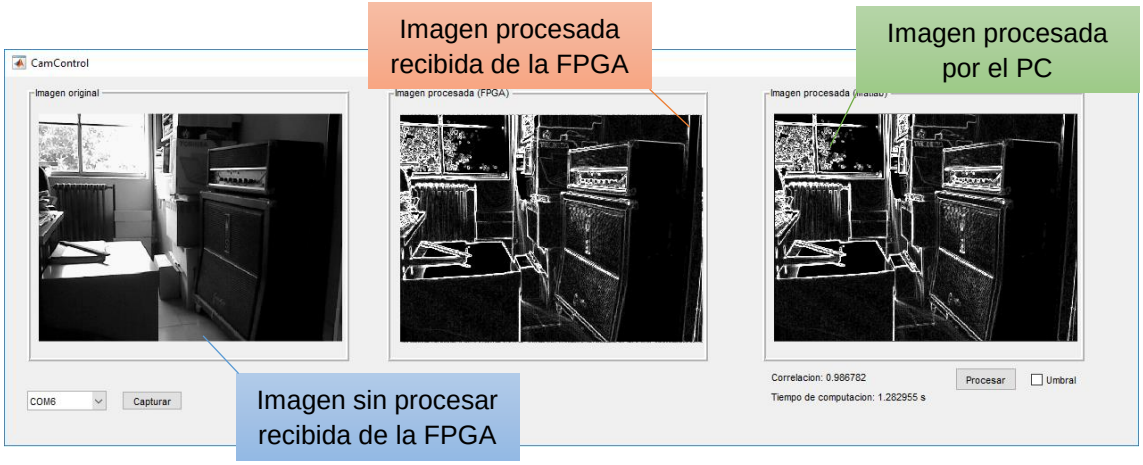

Figura 55. Interfaz gráfica (GUI) de la aplicación Matlab.

Debajo de la tercera ventana se muestra la correlación, entre la imagen procesada por la FPGA y el PC, y el tiempo de computación del procesado por el PC.

Los valores de la correlación obtenidos experimentalmente nunca llegan a ser 1. Esto es debido a que la imagen procesada y enviada por la FPGA es de 640x480 píxeles, donde los bordes no forman parte de la imagen, sino que son valores aleatorios de la memoria DDR. Estos bordes son necesarios para mantener la relación de aspecto de la imagen.

Por otra parte, la imagen resultante del procesado por el PC es de 638x478, debido a que los bordes no necesarios, estos han sido eliminados. Por tanto, el algoritmo de correlación por el hecho de que sean de distinto tamaño es ya menor que 1.

Para contrastar el tiempo de computación, se exponen a continuación las características técnicas del ordenador utilizado experimentalmente:

CPU	Intel Core i7-2630QM @ 2.00 GHz
Memoria	4 GB DDR3 PC3-10700 (667 MHz) CL9
GPU	AMD Radeon HD 6740M
Sistema operativo	Windows 10 Pro 64 bit

Tabla 4. Características técnicas del PC utilizado.

4.3.4. Procedimiento

El primer paso es realizar el montaje del sistema. Para ello es necesario seguir el esquema mostrado en la figura 56.

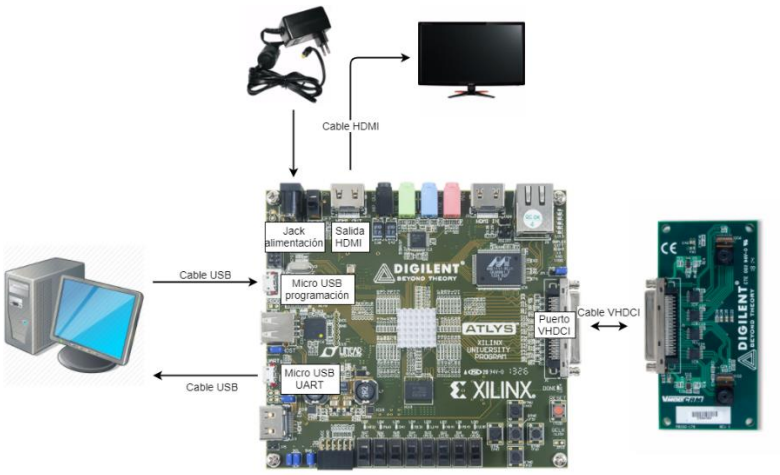

Figura 56. Esquema de montaje del sistema.

Una vez esté listo el montaje, se conmuta el interruptor junto al Jack de alimentación para encender la placa.

Después se cargan los binarios de programación del proyecto a la FPGA a través del software Digilent Adept.

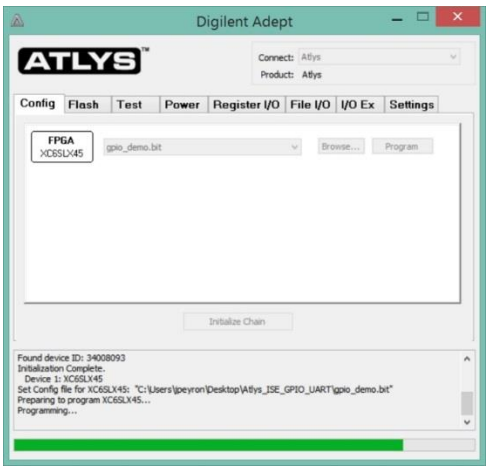

Figura 57. Captura del programa Digilent Adept.

En este momento ya se puede visualizar en el monitor las imágenes proporcionadas por la cámara.

Se puede manipular la visualización del sistema en función de la disposición de los ocho interruptores inferiores, de derecha a izquierda SW0, SW1…SW7 (hacía arriba está activado). A continuación, se describen sus funcionalidades.

- SW0: desactivado detiene el proceso de filtrado, por lo que congela el último fotograma de la imagen procesada.
- SW1: activado muestra en pantalla la imagen filtrada, desactivado muestra la imagen original.
- SW3 y SW2: selección del filtro. Si “01” se selecciona el filtro Emboss, en cualquier otro caso el filtro Sobel.
- SW7: activado añade el nivel de umbral mínimo a la imagen filtrada.

Para realizar la transferencia a PC, hay que abrir Matlab y cargar el programa GUI. Después seleccionar el puerto COM al que está conectada la FPGA. Para ello se hace click derecho en el desplegable de selección, luego se abre el desplegable y se elige.

La transferencia se inicia clicando el botón “Capturar”, el proceso dura unos segundos.

Figura 58. Captura de la GUI al inicializarse.

Una vez completada la transferencia, habrá que clicar el botón “Procesar” para generar y mostrar la imagen filtrada por el PC.

5. CONCLUSIONES

Durante este trabajo se ha desarrollado un sistema complejo basado en FPGA el cual realiza el procesado de imágenes obtenidas a través de una cámara y transmite el resultado a través de una interfaz de video digital de alta velocidad.

Se ha pretendido estudiar y comprender todos los subsistemas que comprenden el trabajo, principalmente interfaz con memoria DDR, transmisión de datos de vídeo digital de alta velocidad y captura de imágenes e interfaz con cámara, los cuales son algunos de los pilares básicos del diseño digital avanzado en la actualidad.

En un principio parecía una tarea sencilla ya que se ha partido de un sistema probado y un funcionando proporcionado por el fabricante de la placa de desarrollo. Pero esto no fue más que el comienzo del trabajo, ya que la comprensión de un diseño VHDL poco documentado puede llegar a ser una tarea difícil incluso para una persona experimentada.

Pero poco a poco, realizando pequeñas modificaciones en el diseño y cargándolo en la placa, se fue consiguiendo mejor comprensión del sistema hasta el punto de poder comenzar la realización del diseño objetivo del trabajo. En este momento no se comprendía el sistema en su totalidad ya que a medida que se avanzaba en el trabajo se fue profundizando cada vez más debido a las necesidades de las modificaciones, que exigían un mayor control del sistema base.

En la siguiente etapa del trabajo se comenzó a estudiar la forma de implementar el algoritmo de detección de bordes, siempre intentando alterar lo menos posible el diseño base. Se optó por un diseño al que se le añadieran los menos bloques posibles y estos fueran reutilizados lo máximo posible, por lo que resultaron cuatro bloques importantes: control de lectura y escritura en memoria, control de la convolución y bloque de convolución.

Cabe mencionar que el diseño claramente no se encuentra optimizado, ya que todo el proceso se realiza en su mayor parte de forma secuencial, a excepción de la convolución que se realiza de forma paralela. Aun así, la FPGA consigue hacer funcionar el sistema en tiempo real para una imagen de 800x480 píxeles. Una propuesta de futuro posible sería optimizar el sistema para aprovechar aún más las capacidades de la FPGA, aumentando la resolución de las imágenes y su consecuente tasa de datos.

Finalmente se concluye que realizar un sistema basado en FPGA es una tarea que requiere más tiempo comparado con el uso de microprocesadores, ya que con estos disponen de multitud de software especializado y su programación es más sencilla. Esto se pudo comprobar cuando se implementó el mismo algoritmo de detección de bordes en Matlab, realizándose en escaso tiempo.

Sin embargo, se ha podido comprobar la potencia de la que dispone una FPGA, siendo capaz de la adquisición de imágenes de una cámara, realizar toda la tarea computacional del sistema y la implantación de una interfaz HDMI, todo en tiempo real.

6. BIBLIOGRAFÍA

Aptina. (2005). *MT9D112 1/4-Inch 2Mp System-On-A-Chip (SOC) CMOS Digital Image Sensor.*

Deutron Electronics Corp. (s.f.). *1G bits DDR2 SDRAM P3R1GE3JGF P3R1GE4JGF.* Hoja de datos.

Digilent. (2011). *VmodCAM Reference Manual.*

Digilent. (2016). *Atlys FPGA Board Reference Manual.*

Digital Display Working Group. (1999). *Digital Visual Interface.* Especificación.

Farooq, U. M. (2012). Chapter 2. FPGA Architectures: An Overview. En *Tree-based Heterogeneous FPGA Architectures.* Springer.

Gonzalez, R. C., & Woods, R. E. (2007). *Digital Image Processing (3rd Edition).* Pearson.

Mendías, J. (2015). Tema 3: El entorno de diseño Xilinx ISE Design Suite. En *Diseño automático de sistemas.*

Xilinx. (2010). *Spartan-6 FPGA Configurable Logic Block (UG384).*

Xilinx. (2010). *Spartan-6 FPGA Memory Controller (UG388).*

Xilinx. (2011). *Spartan-6 Family Overview.*

Xilinx. (2012). *Spartan-6 FPGA Memory Interface Solutions (UG416).*

Zamora, M. M. (2012). *Estudio de un sistema de vision estereo controlado por hardware.* Proyecto Fin de Carrera.

Anexo I: Registros y variables modificadas de la cámara.

- Descripción de los registros modificados:

Reg #	Bits	Default	Name
0x3214	15:0	0x0C80	Pad Slew (RW)
	15:12	X	Reserved
	11	0x0001	Enables power down of VDD GPIO
	10:8	0x0004	Slew rate for PIXCLK 7=fastest slew; 0=slowest. Actual slew depends on load, temperature and I/O voltage. See pad datasheet for details.
	7	0x0001	Input pad power down enable Reserved
	6:4	0x0000	Slew rate for GPIO 7=fastest slew; 0=slowest. Actual slew depends on load, temperature and I/O voltage. See pad datasheet for details.
	3	X	Reserved
	2:0	0x0000	Slew rate for DOUT[7:0] PIXCLK FRAME_VALID and LINE_VALID. 7=fastest slew rate; 0=slowest slew rate. Actual slew rate depends on load, temperature and I/O voltage. See pad datasheet for details.
	7=fastest slew; 0=slowest. ёActual slew depends on load, temperature and I/O voltage. See pad datasheet for details.		
Reg #	**Bits**	**Default**	**Name**
0x341E	15:0	0x8F0B	PLL/ Clk_in control (RW)
	15:12	0x0008	pll_pfd PLL phase detector gain
	11:8	0x000F	0x000F pll_div8_en Independent enables for each of the four identical clock feeds
	7:5	X	Reserved
	4	0x0000	ip_pd Asserted to disable the clock input receiver
	3	0x0001	hyst_en Asserted to enable hysteresis on the clock pin
	2	0x0000	phy_test Asserted to power down PLL and assume normal full chip operation using the external clock as source
	1	0x0001	0x0001 pll_pd PLL power down
	0	0x0001	pll_bypass Asserted to bypass the PLL (for test and low power modes)

Reg #	Bits	Default	Name
0x341C	15:0	0x0150	PLL Dividers1 (RW)
	15:14	X	Reserved
	13:8	0x0001	pll_n_div PLL n-divider value NOTE: pll vco frequency = clkin frequency * pll_m/(pll_n+1)
	7:0	0x0050	pll_m_div PLL m-divider value
Reg #	**Bits**	**Default**	**Name**
0x3202	15:0	0x0009	Standby Control (RW)
	15	RO	SOC standby pin is mapped to this bit
	14:6	X	Reserved
	5	0x0000	Reserved
	4	0x0000	Synchronize standby entry with the end of the frame If bit is set than standby entry is postponed until the end of current frame. If standby request comes between frames standby will be entery will have 1 frame latency.
	3	0x0001	Enable IRQ. When this bit is set IRQ is used to enter standby procedure 2020SOC enters standby mode by means of IRQ. If this bit is set standby functionality is enabled. If this bit is cleared both pin and I2C means of standby entry are disabled. 2020SOC can only enter standby through MCU command in this case.
	2	0x0000	Stop MCU on power up This bit stops MCU and allows host to upload configuration settings
	1	0x0000	Do not initialize variables after standby. If this bit is set most of the memory content will be preserved on power up.
	0	0x0001	Standby SHiP Standby request through SHiP interface. Set High to request Standby Mode.
Reg #	**Bits**	**Default**	**Name**
0x3204	15:0	0x0001	Standby Done Status Bit (RW)
	15:3	X	Reserved
	2	0x0000	Reserved
	1	0x0000	This bit is set to '1' when initial initializateon is done. This bit is set to '1' when initial initializateon is done. Only works if 0x3202[2] is set.
	0	0x0001	This bit will be set upon standby entry Set to '1' one when chip is in a standby. Should be used for host to indicate that it is safe to gate clock.
	This bit indicates that standby procedure has been completed.		

Reg #	Bits	Default	Name
0x338C	15:0	0x0000	Microcontroller variable/RAM address (RW)
	15	0x0000	8-bit access 1=8-bit access; 0=16-bit
	14:13	0x0000	Select Logical Access Bits 14:13 of address for physical access; R0x338C [14:13]=01 select logical access
	12:8	0x0000	Driver ID Bits 12:8 of address for physical access; driver ID for logical access
	7:0	0x0000	Driver variable Bits 7:0 of address for physical access; driver variable offset for logical access
	Microcontroller variables are similar to SHIP registers, except that they are located in MCU memory. Variables are accessed by specifying their address in and reading/writing the value to . Variables can be accessed as 8-bit (byte) and 16-bit (word) at a time. Variable address can be specified as physical or logical. Physical address is the actual address of the variable in the micro-controller's 64K address space. Use physical address to upload custom binary code to a known memory location. Use logical address to configure driver variables. A logical address consists of a driver ID (0=monitor, 1=sequencer, etc) and an offset into the driver data structure.		
Reg #	**Bits**	**Default**	**Name**
0x3390	15:0	0x0000	MCU variable/RAM data (burst 0) (RW)
	To read a variable from MCU memory, set address in R0x338C and read data from Rox3390. To write to a variable, set address in R0x338C and write to R0x3390. 16-bit and 8-bit variables can be accessed, see R0x3390. When reading an 8-bit variable (R0x338C [15]=0) R0x3390 [15:8] are set to 0. When writing to an 8-bit variable, R0x3390 [15:8] are ignored		

Reg #	Bits	Default	Name
0x301A	15:0	0x0248	reset_register (RW)
	15	0x0000	grouped parameter hold 0 = update of many of the registers is synchronized to frame start. 1 = inhibit register updates; register changes will remain pending until this bit is returned to 0. When this bit is returned to 0, all pending register updates will be made on the next frame start.
	14:11	X	Reseved
	10	0x0000	Restart Bad Frames 1 = a restart is forced any time a bad frame is detected. This can shorten the delay when waiting for a good frame, since the delay for masking out a bad frame will be the integration time rather than the full-frame time.
	9	0x0001	Mask Bad Frames 0 = the sensor will produce bad (corrupted) frames as a result of some register changes 1 = bad (corrupted) frames are masked within the sensor by extending the vertical blanking time for the duration of the bad frame.
	8	0x0000	GPI Enable 0 = the primary input buffers associated with the GPI0, GPI1, GPI2, GPI3 inputs are powered down and the GPI cannot be used. 1 = the input buffers are enabled and can be read through Reg0x3026-7.
	7	0x0000	Parallel Enable 0 = the parallel data interface (DOUT[9:0], LINE_VALID, FRAME_VALID, and PIXCLK) is disabled and the outputs are placed in a high-impedance state. 1 = the parallel data interface is enabled. The output signals can be switched between a driven and a high-impedance state using output-enable control See “Streaming/Standby Control” on page 79.
	6	0x0001	Drive Pins 0 = the parallel data interface (DOUT[9:0], LINE_VALID, FRAME_VALID, and PIXCLK) may enter a high-impedance state (depending upon the configuration of Reg0x3026). See “Output-Enable Control” on page 76. 1 = the parallel data interface is driven.
	5	0x0000	Reserved
	4	0x0000	Standby EOF 0 = Transition to standby is synchronized to the end of a sensor row readout (held-off until LINE_VALID has fallen) 1 = Transition to standby is synchronized to the end of a frame.

	3	0x0001	Lock/Unlock Registers Many registers that are specified as read-only are actually implemented as read/write registers. Clearing this bit allows such registers to be written.
	2	0x0000	Start/Stop Streaming Setting this bit places the sensor in streaming mode. Clearing this bit places the sensor in a low-power mode. The result of clearing this bit depends upon the operating mode of the sensor.
	1	0x0000	Restart This bit always reads as 0. Setting this bit causes the sensor to truncate the current frame and start resetting the first row. The delay before the first valid frame is read out equals the integration time.
	0	0x0000	Reset This bit always reads as 0. Setting this bit initiates a reset sequence: the frame being generated will be truncated.

- **Descripción de las variables modificadas:**

ID #	Var #	Bits	Default	Assigned	Name
7	0x095	15:0	0x0000	0x0030	output_format_A (RW)
		15:9	X	0	Reserved
		8	0x0000	0	Turn on processed Bayer mode Chip output data in Bayer fomat. As a result datarate decreses in two times.
		7:6	0x0000	00	RGB output format 00 = 16-bit RGB565 01 = 15-bit RGB555 10 = 12-bit RGB444x 11 = 12-bit RGBx444
		5	0x0000	1	RGB/YUV output 1=output RGB (see R0x332E [7:6]) 0=output YUV
		4	0x0000	1	Use CCIR656 codes when bypassing FIFO 1=use CCIR656 codes when bypassing FIFO 0xAB = frame start 0x80 = line start 0x9D = line end 0xB6 = frame end
		3	0x0000	0	Monochrome output
		2	0x0000	0	Progressive Bayer
		1	0x0000	0	Swaps chrominance byte with luminance byte in YUV output. In RGB mode, swaps odd and even bytes. This bit is subject to synchronous update.
		0	0x0000	0	Swap Channels In YUV output mode, swaps Cb and Cr channels. In RGB mode, swaps R and B. This bit is subject to synchronous update.
		Output Format Config. (context A shadow register). Changes take effect only after REFRESH command.			

ID #	Var #	Bits	Default	Assigned	Name
7	0x019	15:0	0x046C	0x046C	Read Mode A (RW)
		15:14	0x0000	00	Special LINE_VALID 00 = Normal behavior of LINE_VALID 01 = LINE_VALID is driven continuously (continue generating LINE_VALID during vertical blanking) 10 = LINE_VALID is driven continuously as LINE_VALID XOR FRAME_VALID
		13:12	X	00	Reserved
		11	0x0000	0	x bin enable Enable analogue binning in X (column) direction. When set, x_odd_inc must be set to 3 and y_odd_inc must be set to 1.
		10	0x0001	1	xy bin enable Enable analogue binning in X and Y (column and row) directions. When set, x_odd_inc and y_odd_inc must be set to 3.
		9:8	X	00	Reserved
		7:5	0x0003	011	X odd increment Increment applied to odd addresses in X (column) direction. 1= Normal readout 3 = Read out alternate pixel pairs to halve the amount of horizontal data in a frame.
		4:2	0x0003	011	Y odd increment Increment applied to odd addresses in Y (row) direction. 1= Normal readout 3 = Read out alternate pixel pairs to halve the amount of vertical data in a frame.
		1	0x0000	0	Vertical Flip 0 = Normal readout 1 = Readout is flipped (mirrored) vertically so that the row specified by y_addr_end_ is read out of the sensor first. Setting this bit will change the bayer pixel order (see Reg0x3024). The bit-order of bits [1:0] match the order in Reg0x301C but is reversed relative to earlier Micron Imaging sensors.
		0	0x0000	0	Horizontal Mirror 0 = Normal readout 1 = Readout is mirrored horizontally so that the column specified by x_addr_end_ is read out of the sensor first. Setting this bit will change the bayer pixel order (see Reg0x3024).
		context A shadow register R0x20:0. Changes take effect only after REFRESH_MODE command.			

ID #	Var #	Bits	Default	Assigned	Name
7	0x003	15:0	0x0320	0x0280 (640)	Output Width A (RW)
		Output size of final image for context A. Must be equal to or smaller than crop dimension. Changes take effect only after REFRESH command.			
	0x005	15:0	0x0258	0x01E0 (480)	Output Height A (RW)
		Output size of final image for context A. Must be equal to or smaller than crop dimension. Changes take effect only after REFRESH command.			
	0x051	15:0	0x0000	0x0000	crop_X0_A (RW)
		Lower-x scaler zoom window (context A shadow register). Changes take effect only after REFRESH command.			
	0x055	15:0	0x0000	0x0000	crop_Y0_A (RW)
		Lower-y scaler zoom window (context A shadow register). Changes take effect only after REFRESH command.			
	0x053	15:0	0x0320	0x0320 (800)	crop_X1_A (RW)
		Upper-x scaler zoom window (context A shadow register). Changes take effect only after REFRESH command.			
	0x057	15:0	0x0258	0x0258 (600)	crop_Y1_A (RW)
		Upper-y scaler zoom window (context A shadow register). Changes take effect only after REFRESH command.			

- **Secuencia de transferencias por I2C:**

Tipo acceso	Valor transf.	Dirección	Dato	Driver ID	Driver variable	Nombre registro	Nombre variable	Descripción
Read	30000000	3000	0000					Chip version. Default 0x1580
Write	32140D85	3214	0D85			[SOC]Pad Slew		Slew rate control, PCLK 5, D 5
Write	341E8F0B	341E	8F0B			[SOC]PLL/ Clk_in control (RW)		PLL control; Default 0x8F0B
Write	341C0250	341C	0250			[SOC]PLL Dividers1 (RW)		PLL dividers; M=80,N=2,fMCLK=fCLKIN*M/(N+1)/8=80MHz
Write	341E8F09	341E	8F09			[SOC]PLL/ Clk_in control (RW)		PLL control; Power-up PLL; wait 1ms after this!
Write	341E8F08	341E	E8F08			[SOC]PLL/ Clk_in control (RW)		PLL control; Turn off bypass
Write	3202000C	3202	000C			[SOC]Standby Control (RW)		Take camera out of standby, but Stop MCU to allow configuration
Read	32040000	3204	0000			[SOC]Standby Done Status Bit (RW)		Bit [1] should be checked for init status
Write	338C2795	338C	2795	7	0x95	[SOC]Microcontroller variable/RAM address (RW)	output_format_A (RW)	Output format; Context A shadow
Write	33900030	3390	0030			[SOC]MCU variable/RAM data (burst 0) (RW)		
Write	338C2719	338C	2719	7	0x19	[SOC]Microcontroller variable/RAM address (RW)	Read Mode A (RW)	Read mode; Context A
Write	3390046C	3390	046C			[SOC]MCU variable/RAM data (burst 0) (RW)		
Write	338C2703	338C	2703	7	0x03	[SOC]Microcontroller variable/RAM address (RW)	Output Width A (RW)	Output width; Context A

Tipo acceso	Valor transf.	Dirección	Dato	Driver ID	Driver variable	Nombre registro	Nombre variable	Descripción
Write	33900280	3390	0280			[SOC]MCU variable/RAM data (burst 0) (RW)		640
Write	338C2705	338C	2705	7	0x05	[SOC]Microcontroller variable/RAM address (RW)	Output Height A (RW)	Output height; Context A
Write	339001E0	3390	01E0			[SOC]MCU variable/RAM data (burst 0) (RW)		480
Write	338C2751	338C	2751	7	0x51	[SOC]Microcontroller variable/RAM address (RW)	crop_X0_A (RW)	Crop X0; Context A
Write	33900000	3390	0000			[SOC]MCU variable/RAM data (burst 0) (RW)		0
Write	338C2755	338C	2755	7	0x55	[SOC]Microcontroller variable/RAM address (RW)	crop_Y0_A (RW)	Crop Y0; Context A
Write	33900000	3390	0000			[SOC]MCU variable/RAM data (burst 0) (RW)		0
Write	338C2753	338C	2753	7	0x53	[SOC]Microcontroller variable/RAM address (RW)	crop_X1_A (RW)	Crop X1; Context A
Write	33900320	3390	0320			[SOC]MCU variable/RAM data (burst 0) (RW)		800
Write	338C2757	338C	2757	7	0x57	[SOC]Microcontroller variable/RAM address (RW)	crop_Y1_A (RW)	Crop Y1; Context A
Write	33900258	3390	0258			[SOC]MCU variable/RAM data (burst 0) (RW)		600
Write	32020008	3202	0008			[SOC]Standby Control (RW)		Kick-start MCU
Write	301A02CC	301A	02CC			[Core]reset_register (RW)		reset/output control; parallel enable, drive pins, start streaming