



Universidad Politécnica de Madrid

Departamento de
Ingeniería
Electrónica

UNIVERSIDAD POLÍTÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN INGENIERÍA DE SISTEMAS ELECTRÓNICOS

TRABAJO FIN DE MÁSTER

**Implementación de un sistema de
filtrado de imagen sobre FPGA y
proyección mediante interfaz VGA**

Germán Bravo López

Madrid, 2019

TRABAJO FIN DE MÁSTER

TÍTULO: Implementación de un sistema de filtrado de imagen sobre FPGA y proyección mediante interfaz VGA

AUTOR: Germán Bravo López

TUTOR: Juan Antonio López Martín

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

MIEMBROS DEL TRIBUNAL CALIFICADOR:

FIRMA

PRESIDENTE:

VOCAL:

SECRETARIO:

SUPLENTE:

FECHA DE LECTURA:

CALIFICACIÓN:

A mis padres, a mi hermano y a Elia. A mi abuelo, por estar siempre a mi lado y darme todo su apoyo.

A los compañeros y amigos que he tenido la suerte de conocer durante este Máster. También agradecer a todos mis profesores por la formación recibida.

Resumen

En este proyecto se ha realizado la implementación de un sistema de filtrado de imagen utilizando un lenguaje de descripción de hardware, VHDL. El sistema aquí descrito presenta un método de captación de imágenes mediante un sensor de imagen CMOS, concretamente el OV7670, que se conecta a una FPGA mediante la interfaz *Pmod™*, estándar abierto de comunicación definido por *Digilent® Inc.* Además, dispone de un subsistema que permite la comunicación mediante VGA con un monitor CRT, para la proyección de los resultados obtenidos durante el proceso de filtrado.

Inicialmente, se ha realizado una contextualización y explicación detallada de los materiales de los que se disponen y de los funcionamientos de cada uno de ellos, además de los principios matemáticos de los procedimientos utilizados en el sistema.

Posteriormente, tras el diseño de cada módulo, se ha realizado una comprobación del diseño y la implementación mediante una serie de tests sobre los diferentes módulos, que permitían garantizar un correcto comportamiento de los sistemas de forma independiente; en función de la tarea de cada uno. Sobre el sistema implementado, se han incluido una serie de *kernels* o máscaras de filtrado convolucional, cuyos resultados se han ilustrado y discutido.

La implementación se ha realizado sobre la placa de desarrollo *Nexys 4 DDR™* de la empresa *Digilent Inc.*, la cual tiene en su interior una FPGA *Artix®-7* de *Xilinx®*.

Palabras clave

VHDL, FPGA, Xilinx Artix-7, sensor OV7670 CMOS, VGA, *Pmod Interface*, protocolo SCCB, filtrado de imagen, convolución, MATLAB.

©Germán Bravo López. Madrid - 2019 - Todos los derechos reservados

©Universidad Politécnica de Madrid - 2019

Esta obra está bajo una licencia [Creative Commons “Reconocimiento-NoCommercial-CompartirIgual 3.0 España”](#).



Abstract

In this project, an image filtering system has been implemented using a hardware description language, VHDL. The system described here presents an image capture method using a CMOS image sensor, specifically the OV7670, which connects to an FPGA through the *Pmod™* interface, an open standard defined by *Digilent® Inc.* In addition, it has a subsystem that allows communication via VGA with a CRT monitor, for the projection of the results obtained from the filtering process.

First of all, a contextualization and a detailed explanation of the materials available and the functioning of each of them has been carried out, as well as the mathematical principles of the procedures used within the system.

Subsequently, after the design of each module, a verification of the design and the implementation has been made by means of a series of tests on the different modules, which allowed to guarantee the correct behavior of the systems in an independent way; on behalf of each one's task. On the implemented system, a series of *kernels* or convolutional filter masks have been included, the results of them have been illustrated and discussed.

The implementation has been done on the development board *Nexys 4 DDR™* of *Digilent Inc.* company, which has in it an *Artix®-7* FPGA of *Xilinx®*.

Keywords

VHDL, FPGA, Xilinx Artix-7, OV7670 CMOS sensor, VGA, *Pmod Interface*, SCCB communication protocol, image filtering, convolution, MATLAB.

Índice general

Agradecimientos	V
Resumen	VII
1 Introducción	1
1.1 Objetivos y motivación	2
2 Antecedentes y estado del arte	5
2.1 FPGA	5
2.1.1 Mercado de FPGA	7
2.1.2 Sectores de aplicación	8
2.2 Procesamiento de imagen	10
3 Materiales y fundamentos teóricos	13
3.1 Placa de desarrollo <i>Nexys 4 DDR</i>	13
3.1.1 FPGA <i>Artix-7</i>	15
3.1.2 Interfaz <i>Pmod</i>	18
3.1.3 Interfaz VGA	19
3.1.3.1 Funcionamiento de un monitor de rayos catódicos	20
3.2 Sensor de imagen CMOS	21
3.2.1 OV7670 CameraChip™	22
3.3 Filtrado de imagen	28
3.3.1 Filtros de convolución	29
3.3.2 Bordes de la imagen	31

4 Arquitectura del sistema	35
4.1 Sistema de comunicaciones con el OV7670	36
4.1.1 Bloque de captura	36
4.1.2 Bloque de memoria	37
4.1.3 Bloque de control	39
4.1.3.1 Bloque emisor SCCB	40
4.1.3.2 Bloque de registros	42
4.2 Sistema de filtrado	43
4.3 Sistema controlador de VGA	47
5 Resultados y Discusión	51
5.1 Resultados de los kernels	51
5.2 Test y cronogramas	55
5.3 Recursos	58
5.4 Consumo de potencia	60
6 Conclusiones	61
7 Líneas futuras	63
A Aspectos éticos, sociales, económicos y ambientales	65
A.1 Impactos ético-sociales	65
A.2 Impactos económicos	66
A.3 Impactos ambientales	66
B Presupuesto económico	69
C Resultados sobre una imagen a color	71
Bibliografía	75

Índice de figuras

1.1 Ejemplo de los resultados de la localización del hipocampo en datos reales. (a) Imagen original (b) Contorno original (c) Contorno final (d) Región detectada	2
2.1 Tabla comparativa de las Familias más importantes de Xilinx [12, 13].	7
2.2 Cinta de transmisión del sistema Bartlane codificada en código Baudot.	10
2.3 Cámara digital.	11
3.1 Placa de desarrollo Nexys 4 DDR [17].	14
3.2 Tabla comparativa de los modelos de la familia Artix-7.	16
3.3 Arquitectura de la serie Artix-7 [14].	17
3.4 Layout de la Serie-7 de FPGAs de Xilinx [12].	18
3.5 Pines de la interfaz Pmod.	19
3.6 Pines de conexión de la interfaz VGA.	19
3.7 Pantalla CRT de color.	20
3.8 Sincronización horizontal del haz de electrones	21
3.9 Arquitectura interna de un sensor de imagen CMOS.	22
3.10 Diagrama de bloques del SoC OV7670.	23
3.11 Esquemático de la placa OV7670 que se ha adquirido para el proyecto.	24
3.12 Fotografía de la placa OV7670 que se utiliza en este proyecto.	25
3.13 Temporización horizontal de los datos de salida.	26
3.14 Temporización de los datos de cada fotograma.	26
3.15 Proceso de convolución [28].	29
3.16 Ejemplo de recorte (<i>cropping</i>).	32

3.17 Ejemplo de extensión de frontera (<i>extended border</i>).	32
3.18 Ejemplo de relleno de ceros (<i>zero-padding</i>).	32
3.19 Ejemplo de envolvente (<i>wrap-around</i>).	33
4.1 Diagrama de bloques ilustrativo del sistema completo.	35
4.2 Bloque de captura de imagen.	37
4.3 Bloque de memoria.	38
4.4 Bloque de control del OV7670.	39
4.5 Inicio de la transmisión en SCCB de 2 cables.	40
4.6 Estructura de un mensaje en SCCB de 2 cables.	41
4.7 Fin de la transmisión en SCCB de 2 cables.	41
4.8 Esquema de interconexión de cada uno de los bloques del sistema de captura de imagen y configuración del sensor OV7670.	42
4.9 Bloque de filtrado de imagen.	43
4.10 Ilustración que ejemplifica los valores de las direcciones en RAM de cada píxel.	44
4.11 Método de carga de los datos desde la memoria RAM.	44
4.12 Máquina de estados de la salida de color sobre la pantalla.	45
4.13 Gráfico del flujo de datos para la implementación de Ec. 3.2.	46
4.14 Diagrama de bloques internos del funcionamiento del controlador VGA. . . .	47
4.15 Ilustración de las señales de sincronización VGA sobre una pantalla.	49
5.1 Imagen pura de resolución 640x480.	52
5.2 Resultado del filtro con el kernel 1 (detección de bordes 1) sobre Fig. 5.1. . .	52
5.3 Resultado del filtro con el kernel 2 (detección de bordes 2) sobre Fig. 5.1. . .	53
5.4 Resultado del filtro con el kernel 3 ('Sobel' horizontal) sobre Fig. 5.1. . . .	53
5.5 Resultado del filtro con el kernel 4 ('Sobel' vertical) sobre Fig. 5.1.	53
5.6 Resultado del filtro con el kernel 5 (enfoque) sobre Fig. 5.1.	54
5.7 Resultado del filtro con el kernel 6 (suavizado) sobre Fig. 5.1.	54
5.8 Resultado del filtro con el kernel 7 (filtro Norte) sobre Fig. 5.1.	54
5.9 Resultado del filtro con el kernel 8 (filtro Este) sobre Fig. 5.1.	55

5.10 Imagen ampliada de Fig. 5.1.	56
5.11 Imagen ampliada de Fig. 5.1 tras el filtrado con el kernel 3.	56
5.12 Resultado en Vivado del proceso de filtrado de la esquina superior izquierda de Fig. 5.1.	57
5.13 Resumen de la utilización de recursos en la FPGA para el sistema completo.	59
5.14 Resumen del consumo de potencia en la FPGA.	60
C.1 Imagen a color de resolución 640x480.	71
C.2 Resultado del filtro con el kernel 1 sobre Fig. C.1.	72
C.3 Resultado del filtro con el kernel 2 sobre Fig. C.1.	72
C.4 Resultado del filtro con el kernel 3 sobre Fig. C.1.	72
C.5 Resultado del filtro con el kernel 4 sobre Fig. C.1.	73
C.6 Resultado del filtro con el kernel 5 sobre Fig. C.1.	73
C.7 Resultado del filtro con el kernel 6 sobre Fig. C.1.	73
C.8 Resultado del filtro con el kernel 7 sobre Fig. C.1.	74
C.9 Resultado del filtro con el kernel 8 sobre Fig. C.1.	74

Índice de tablas

2.1	Comparación entre FPGA y ASIC [9, 10].	6
3.1	Componentes que contiene la placa de desarrollo Nexys 4 DDR.	14
3.2	Formato RGB565.	27
3.3	Formato RGB555.	27
3.4	Formato RGB444.	27
4.1	Tabla de tiempos y equivalencias en los contadores para VGA.	48
5.1	Tabla de utilización de recursos totales.	58
5.2	Tabla de utilización de recursos por módulo.	59

Capítulo 1

Introducción

La visión es un sentido mediante el cual el ser humano percibe grandísima cantidad de información todos los días. Esta información es procesada y analizada en el cerebro para interpretar resultados en cualquier situación de la vida, como por ejemplo cuando dos personas mantienen una conversación y solo con la mirada analizan y entienden el estado de ánimo de la otra persona, o para comprobar si está lloviendo a través de la ventana antes de salir y así saber si hay que coger el paraguas. Si se tiene en cuenta que en muchas circunstancias el ser humano no puede intervenir u observar directamente con su mirada alguna situación que desea analizar (en el espacio, en operaciones quirúrgicas o radiografías, en el fondo marino, etc.), se llega a la conclusión de que se necesitan instrumentos que capturen imágenes y las procesen de forma adecuada para poder analizarlas y no perder información durante ese proceso [1].

Las aplicaciones en las que resulta fundamental hoy día el procesamiento de imágenes son muy diversas y teniendo en cuenta que, si se realiza con la finalidad de realizar predicciones, las áreas de aplicación aumentan en mayor medida. En el área de la medicina, por ejemplo, se utilizan numerosos procesos de reducción del ruido de las imágenes que se obtienen para poder realizar diagnósticos de forma adecuada. Aquí es crítico no cometer errores provocados por un mal diagnóstico o una mala interpretación de los resultados obtenidos de imágenes simuladas o fotografiadas. Es vital que las herramientas que se utilicen, garanticen una elevada fiabilidad y aporten ayuda.

Cuando se tratan imágenes degradadas por el ruido se hace más laboriosa la tarea de interpretación de la enfermedad o el problema. Aún más, si se utilizan técnicas de diagnóstico asistidas por computador o de clasificación mediante el uso de Inteligencia Artificial, aumentarían los errores cometidos mediante su uso. Por ello, se hace imprescindible la tarea del preproceso y acondicionamiento de la imagen [2, 3].

Un caso muy estudiado es el de la enfermedad del Alzheimer, que se trata de una demencia neurodegenerativa en la que se hace muy compleja la distinción entre un sujeto que la sufre y otro que no.

En Fig. 1.1 se observan los resultados de un método propuesto en [2] que consiste en la

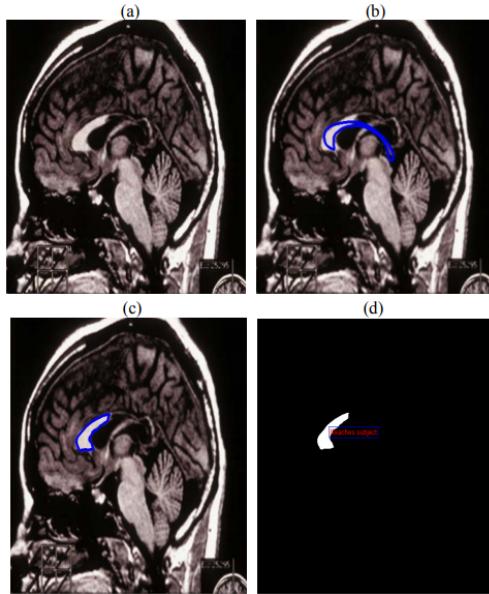


Fig. 1.1: Ejemplo de los resultados de la localización del hipocampo en datos reales. (a) Imagen original (b) Contorno original (c) Contorno final (d) Región detectada

localización del hipocampo sobre imágenes cerebrales y la posterior clasificación de estas en comparación con una base de datos de imágenes afectadas y otras no afectadas, de forma que se permita estimar un posible reconocimiento de la enfermedad.

1.1. Objetivos y motivación

Tras haber mencionado este caso en particular, se puede llegar a la conclusión de que resulta muy importante que existan herramientas capaces de realizar ese preprocesado del que se ha hablado con anterioridad. De ahí, para este trabajo Fin de Máster surge la necesidad de implementar un sistema de filtrado de imagen sobre una FPGA, así como de la proyección de los resultados mediante interfaz VGA.

Un punto importante que hay que destacar cuando se analiza un circuito hardware y se compara con un programa software es que la implementación hardware trabaja directamente sobre el flujo de datos. Con esto se quiere decir que un circuito implementado en hardware realiza una aplicación específica y permite cumplir plazos de tiempo real¹, es decir, no tiene que interpretar una serie de instrucciones o códigos para realizar la tarea que dichas instrucciones interpretan o significan, simplemente es una arquitectura dedicada a una función en particular y por tanto existe en ella un flujo de datos que circula desde unas entradas a unas salidas sin necesidad de esperar a ningún análisis de códigos de funcionamiento.

El objetivo principal de este proyecto es ampliar el conocimiento en el diseño hardware y

¹Para este proyecto se considera este concepto de tiempo real como el procesado de 30 imágenes por segundo.

procesamiento de datos, al mismo tiempo que realizar un estudio de los distintos algoritmos de procesamiento de imagen que existen actualmente. Debido a que el procesamiento de imágenes es un cálculo que requiere una gran capacidad de cómputo, en su mayoría paralelo sobre cada píxel, se utilizan las FPGAs o las GPUs ya que permiten sacar rendimiento a esa paralelización; no como ocurre en las CPUs, ya que estas procesan los datos de manera secuencial y se pierde la oportunidad de explotar esa paralelización. Las FPGAs permiten generar un circuito a nivel hardware, específico para el cálculo que se desea realizar y así poder acelerar el cómputo de los algoritmos que se empleen. De esta forma, pueden ser utilizadas en sistemas con restricciones de tiempo real [4].

Para la realización de este sistema se plantea una serie de objetivos principales, los cuales son necesarios para obtener el funcionamiento completo deseado:

- Estudio y comprensión de la interfaz VGA de conexión entre la FPGA y un monitor de pantalla, con la posterior implementación de dicho módulo para la representación de los resultados.
- Estudio de distintos algoritmos de filtrado de imagen, con el posterior diseño e implementación de estos sobre la FPGA.
- Estudio y compresión del funcionamiento de los sensores de imagen con la finalidad de instalar una cámara para capturar fotografías y comprobar la aplicación de un sistema completo, haciendo uso de la interfaz Pmod² de conexión con la FPGA.
- Optimización de procesamiento del sistema final debido a la sobrecarga de cómputo que origina el tratamiento de imágenes.

Estos objetivos sirven de metodología de trabajo, es decir, permiten seguir un guión de desarrollo enfocado en cada punto. Para situar el proyecto, en el Capítulo 2 se plantea el estado del arte y el contexto en el que se encuentran tanto la tecnología que se va a utilizar como las diferentes formas de filtrado de imagen. En el Capítulo 3 se describen, de forma detallada, los materiales de los que se disponen para el proyecto y los fundamentos teóricos en los que se basan los filtros aplicados. Posteriormente, en el Capítulo 4, se explica en profundidad el diseño propuesto para la implementación del sistema. Los resultados obtenidos y las características del mismo se analizan y discuten en el Capítulo 5. Finalmente, en los capítulos 6 y 7 se establecen una serie de conclusiones extraídas y las líneas futuras propuestas. Además, se incluyen tres apéndices finales: en el Apéndice A se realiza un análisis de los aspectos éticos, económicos, sociales y ambientales del proyecto, en el Apéndice B se incluye una tabla de presupuestos del proyecto y en el Apéndice C se muestran una serie de resultados del sistema de filtrado.

²Siglas de Interfaz de Módulo Periférico (del inglés *Peripheral Module Interface*) es un estándar definido por Digilent Inc. en [5] para periféricos usados con FPGAs o placas de desarrollo de microcontroladores.

Capítulo 2

Antecedentes y estado del arte

Para situar el proyecto en torno a un contexto y dar un marco de aplicación entre los sectores comerciales, el Capítulo 2 pretende servir de resumen de la situación actual del mercado de las FPGAs y los principales sectores en los que se aplica su utilización, además de las técnicas de filtrado que existen y las aplicaciones que estas tienen.

2.1. FPGA

Una FPGA (del inglés *field-programmable gate array*) es un circuito integrado configurable que contiene bloques de lógica en los que se programa tanto la interconexión como la funcionalidad mediante un lenguaje de descripción de hardware [6, 7].

Las FPGAs son actualmente los componentes más representativos de los dispositivos lógicos programables, con características que pueden ser modificadas, manipuladas o almacenadas mediante programación aun después de haber sido puestos en funcionamiento en algún dispositivo. La configuración de una FPGA es generalmente especificada usando un lenguaje de descripción de hardware (HDL) como los usados para el diseño de los ASIC³, tal como puede ser el VHDL o Verilog. Estos dispositivos pueden ser usados para implementar cualquier función lógica que un ASIC puede desempeñar, pero con la habilidad de poder ser reprogramados después de su instalación y con un coste menor del ciclo de prueba. Esto último se debe a que el mismo dispositivo puede ser reprogramado sin necesidad de fabricar uno nuevo, lo que ofrece muchas ventajas en diversas aplicaciones.

Si se comparan las FPGAs con tecnologías similares, se encuentran las siguientes diferencias [8]:

- **Circuitos integrados de aplicación específica (ASIC):** a lo largo de la historia, las FPGAs han sido más lentas, menos eficientes en cuanto al consumo de energía y

³Del inglés *Application-Specific Integrated Circuit*, se trata de un circuito fabricado a medida de una aplicación particular, totalmente lo contrario de los circuitos de propósito general.

generalmente de un menor rendimiento que los ASICs. Sin embargo, las ventajas de usar FPGAs es que tienen un ciclo más corto de desarrollo y fabricación, además de la habilidad de poder ser reprogramados para corregir fallos o incluir mejoras con un menor costo de ciclo de diseño. Algunos fabricantes hacen uso de ambas tecnologías, al desarrollar y probar sus diseños en FPGAs para construir después la versión final de manera que pueda ser modificada después de construida.

- **Dispositivo lógico complejo programable (CPLD):** las diferencias principales entre los CPLD y las FPGA son a nivel arquitectural. Un CPLD está conformado por una estructura consistente en uno o más arreglos programables de suma de productos más o menos restringida cuyo bloque de salida son unos pocos registros controlados por una señal de reloj. Esto trae como resultado una flexibilidad limitada, con la ventaja de que los retrasos de propagación pueden ser estimados de manera más precisa. Por otra parte, en la arquitectura de las FPGA el elemento dominante son las interconexiones. Esto las hace dispositivos mucho más flexibles en cuanto al rango de diseños cuya implementación resulta práctica en ellas, pero es más complejo realizar el diseño de los dispositivos. Otra diferencia remarcable es la presencia en la mayoría de las FPGA de funciones embebidas de alto nivel, como multiplicadores y sumadores, así como también memorias y bloques lógicos para implementar decodificadores o funciones matemáticas.

Como se ha comentado, las FPGAs se utilizan en aplicaciones similares a los ASIC y, a pesar de ser más lentas y tener mayor consumo de energía, se han convertido en un elemento de desarrollo ideal para la realización de prototipos digitales, tal y como se muestra en la Tabla 2.1. Esto se debe a la característica esencial para reducir el *time to market*: la reprogramabilidad, lo que les permite que se puedan actualizar las distintas funcionalidades del sistema o corregir errores detectados de forma mucho más rápida. De este modo, garantizan que los costes de desarrollo sean mucho menores para pequeñas tiradas de dispositivos.

Tabla 2.1: Comparación entre FPGA y ASIC [9, 10].

	FPGA	ASIC
Time to Market	Rápido	Lento
NRE	Bajo	Alto
Flujo de diseño	Simple	Complejo
Coste unitario*	Alto	Bajo
Rendimiento	Medio	Alto
Consumo de potencia	Alto	Bajo
Tamaño de unidad	Medio	Bajo

* Teniendo en cuenta elevados volúmenes de compras.

Las FPGAs han sido conocidas como la base de la creación de prototipos y de la computación reconfigurable. La idea de la informática reconfigurable se remonta a los años sesenta, cuando Gerald Estrin introdujo los primeros conceptos con procesador y matriz de hardware reconfigurable. Desde este momento, la industria de los semiconductores no ha visto nada que

pueda igualar a un microprocesador o FPGA en términos de versatilidad y heterogeneidad de potenciales hasta la actual llegada de los [ACAP](#) (Adaptive Compute Acceleration Platform).

La tecnología de las FPGAs es capaz de explotar el paralelismo hardware y aporta flexibilidad y rapidez en el prototipado. Además, permite un coste menor de Ingeniería No Recurrente (NRE - No Recurring Engineering) en comparación con las ASICs, es decir, reduce el coste de investigación, desarrollo, diseño y prueba de un nuevo producto (Tabla 2.1). Sin embargo, a pesar de tener una inmensa capacidad de configuración y de la elevada flexibilidad que ofrecen, se consideran difíciles de ser adoptadas en algunos productos debido al elevado precio del silicio si se compara con un microprocesador. Por otra parte, el funcionamiento en paralelo con velocidades de reloj más bajas permite reducir el consumo.

2.1.1. Mercado de FPGA

En la actualidad las dos grandes empresas líderes en el mercado de las FPGAs son, con gran diferencia, Xilinx Inc. e Intel (Altera). Tras ellas se sitúan Lattice Semiconductor, Microsemi Corporation (Actel), QuickLogic, Achronix, Teledyne e2v, Nallatech y algunas más [11].

Tan solo centrándose en Xilinx, se puede encontrar una extensa variedad de series y familias de FPGAs con diferentes aplicaciones. En particular, la última serie, la Serie 7, está compuesta por cuatro familias de FPGA que tratan de abarcar el rango completo de posibles requisitos del sistema: desde aplicaciones de bajo coste, pequeño factor de forma, costo-sensibles y alto volumen de aplicaciones, hasta aplicaciones con un enorme ancho de banda de conectividad, capacidad lógica y capacidad de procesamiento de señal para el máximo número de aplicaciones de altas prestaciones [12].

Maximum Capability	ARTIX®	KINTEX®	VIRTEX®
Logic Cells	20K – 355K	70K – 480K	285K – 2,000K
Block RAM	12 Mb	34 Mb	65 Mb
DSP Slices	40 – 700	240 – 1,920	700 – 3,960
Peak DSP Perf.	504 GMACs	2,450 GMACs	5,053 GMACs
Transceivers	4	32	88
Transceiver Performance	3.75Gbps	6.6Gbps and 12.5Gbps	12.5Gbps, 13.1Gbps and 28Gbps
Memory Performance	1066Mbps	1866Mbps	1866Mbps
I/O Pins	450	500	1,200
I/O Voltages	3.3V and below	3.3V and below 1.8V and below	3.3V and below 1.8V and below

Fig. 2.1: Tabla comparativa de las Familias más importantes de Xilinx [12, 13].

La tabla comparativa de Fig.(2.1) muestra un resumen de las características principales

de las 3 familias más importantes de Xilinx, estas se incluyen (junto con la cuarta familia) de forma resumida en la siguiente lista [12, 13, 14].

- Familia *Spartan®-7*: Optimizada para bajo coste, baja potencia y altas prestaciones de E/S. Tecnología de fabricación de 28 nm.
- Familia *Artix®-7*: Optimizada para aplicaciones de baja potencia que requieren transceptores serie y grandes DSP con elevado throughput⁴. Tecnología de fabricación de 28 nm.
- Familia *Kintex®-7*: Optimizada para la mejor relación rendimiento-precio con el doble de mejoras en comparación con la generación anterior. Tecnologías de fabricación de 28 nm, 20 nm (*UltraScale™*), 16 nm (*UltraScale+*).
- Familia *Virtex®-7*: Tecnologías de fabricación de 28 nm, 20 nm (*UltraScale*), 16 nm (*UltraScale+*).

La tecnología *UltraScale+* de Xilinx es la tecnología de mayor rendimiento e integración de Xilinx, se sitúa en la gama más alta de productos.

2.1.2. Sectores de aplicación

El mercado de las FPGAs está creciendo de forma muy rápida hoy en día debido a su amplia integración en numerosos sectores industriales y comerciales (smartphones, automoción, telecomunicaciones). Según un informe de un estudio reciente publicado por Market Research Future [11], el mercado global de las FPGAs está en auge y se espera que gane importancia en el futuro. Se prevé que el mercado muestre un crecimiento espectacular en 2022, superando sus anteriores récords de crecimiento en términos de valor con una sorprendente CAGR⁵ durante el periodo previsto (2016 - 2022).

Como ya se ha mencionado anteriormente, estos dispositivos poseen unas características muy destacables que los diferencian de otros dispositivos programables, aumentando así su utilización en distintos mercados. Por ejemplo, se incluyen en áreas como el procesamiento digital de señales, en el sector aeroespacial y de defensa, en una serie de áreas de aplicación en red (empresas financieras, telecomunicaciones, etc.) y en el sector automóvil relacionado con el control GPS y la visualización 3D para suplir la alta demanda de ADAS (Advanced Driver Assistance System) [14]. También tienen otras aplicaciones como la visión por computador, el reconocimiento del habla, la radioastronomía y algunas otras en las que su aplicación se encuentra en crecimiento, como en los Data Centers, en la codificación y encriptación o en el tratamiento de imágenes biomédicas (Capítulo 1).

⁴Velocidad de salida de datos o velocidad de procesamiento de un programa, es el volumen de trabajo o de información neto que fluye a través de un sistema.

⁵Tasa de crecimiento anual compuesta, del inglés *Compound Annual Growth Rate*, es un término de negocios que se utiliza para describir el crecimiento, sobre un periodo de tiempo, de ingresos u otros elementos de negocio.

En función del sector de aplicación, los desarrolladores de FPGAs crean sistemas inteligentes, conectados y diferenciados, a través de la integración entre los niveles más altos de inteligencia actuales basados en software y la optimización de hardware. Por ejemplo, en la industria aeroespacial y de defensa se precisa continuamente de soluciones IP avanzadas, con dispositivos FPGA y SoC, para reducir el riesgo de la misión y el coste del sistema. Los entornos en los cuales se trabaja en este sector son más exigentes y precisan de dispositivos capaces de actuar correctamente a un amplio rango de temperaturas, además de poseer un alto grado de resistencia.

Por otra parte, el sector automovilístico es uno de los más explotados en la actualidad, donde en los últimos años se ha experimentado un crecimiento considerable enfocado a la conducción autónoma. Las FPGAs juegan un papel importante en la implementación de aplicaciones como los ADAS, sistemas electrónicos que ayudan al conductor a través de una interfaz hombre-máquina. Los principales objetivos de la electrónica en el campo del automóvil son la integración, el control de alta precisión y las comunicaciones visuales con el humano. De esta forma, los ADAS ofrecen una solución escalable e integrada que puede ser implementada mediante la utilización de un dispositivo de este tipo enfocado al control de las cámaras y sensores del sistema. Asimismo, las FPGAs ofrecen bajo consumo de energía permitiendo una funcionalidad segura en la nueva era de la conducción autónoma, protegiendo la información y garantizando que los productos sean tolerantes a fallos.

Como ya se introdujo, otra aplicación de las FPGAs es ofrecer soluciones para imágenes médicas, diagnósticos y equipos clínicos, es decir, responder a las crecientes necesidades de plataformas sanitarias escalables con multiprocesamiento heterogéneo, flexibilidad de E/S (entrada/salida), controles determinísticos basados en hardware y soluciones integrales de ciberseguridad, seguridad y aprendizaje de máquinas. Esto es esencial en campos como la cirugía, donde las soluciones deben de ser lo más precisas posibles para que la aplicación sea mínimamente invasiva sobre el paciente y no se limite la extracción de datos. A esto están ligados los sistemas de visión avanzados que permiten control multiaxial y que ofrecen un procesamiento de imagen de alta resolución en tiempo real y, por tanto, un mejor diagnóstico.

Las aplicaciones anteriormente comentadas son únicamente ejemplos de sectores a los cuales están destinados dispositivos como las FPGAs. En el campo industrial se utilizan para aplicaciones robóticas y de automatización a través del control de sensores inteligentes, plataformas modulares basadas en visión y monitorización en tiempo real.

Además, las FPGAs y los SoCs son muy utilizados en comunicaciones tanto inalámbricas como cableadas. En particular, las redes inalámbricas están evolucionando rápidamente hacia redes heterogéneas que emplean muchos tamaños o clases de soporte, empleando diferentes clases de células de construcción, por lo que los requisitos difieren significativamente en función del rendimiento, el ancho de banda de la señal, los niveles de potencia y el número de usuarios. Esto, utilizando los dispositivos mencionados, se puede implementar en un único dispositivo mediante la reutilización de IP (Intellectual Property) Cores y algoritmos, consiguiendo así comunicaciones inalámbricas rápidas y rentables.

2.2. Procesamiento de imagen

El procesamiento digital de imágenes es el conjunto de técnicas que se aplican sobre las imágenes digitales con el objetivo de obtener una imagen más adecuada para la aplicación específica en la que se utilice.

Se puede decir que los primeros intentos de manipulación, almacenamiento y transmisión de imágenes surgieron en 1920 con la invención de una técnica de transmisión de imágenes codificadas a través de líneas de cable submarino entre Londres y Nueva York. Se trata del sistema Bartlane [15], que utiliza las máquinas de escribir telegráficas para codificar y decodificar la imagen que se envía. Dicha máquina recibe ese nombre por sus inventores, Harry G. Bartholomew y Maynard D. McFarlane. Este sistema se utilizó por primera vez en 1921 y su patente fue concedida en 1927. Permitía transmitir 1 imagen a través del Atlántico, en menos de 3 horas y codificada con 5 niveles de grises (en 1929 ya se hacía con 15).

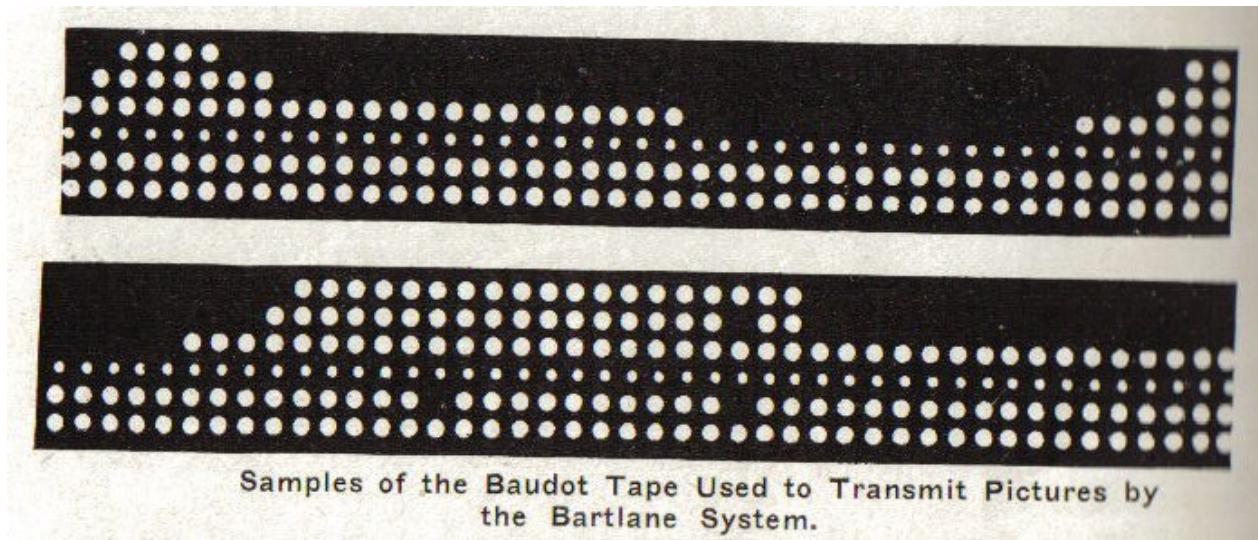


Fig. 2.2: Cinta de transmisión del sistema Bartlane codificada en código Baudot.

Sin embargo, no fue hasta 1957 que no se transmitió por primera vez una imagen digital. En este año Russell A. Kirsch y su equipo de trabajo enviaron una imagen de 176x176 píxeles a través de la computadora SEAC (Standard Eastern Automatic Computer) mediante el escáner que habían inventado. Tecnología que se mejoraría para ser usada por la NASA en la década siguiente para la transmisión de imágenes de teledetección.

Estos dos primeros hechos constituyeron una base sobre la que fundamentar, construir e investigar los diferentes métodos y procesos de tratamiento de imagen hasta conseguir llegar a las formas de trabajo actuales y disponer de las tecnologías para ello. Históricamente la aplicación más común de técnicas de procesamiento digital de imagen es la fotografía de imágenes espaciales y su envío desde satélites. Los descubrimientos que se iban realizando se utilizaban para crear nuevas formas de procesamiento y expandir el área o los ámbitos de aplicación de las mismas.

El procesamiento digital de imágenes es un área en la cual se está investigando actualmente y se van obteniendo nuevas técnicas con aplicaciones muy interesantes. El objetivo es, a partir de una imagen, obtener otra modificada mediante la utilización de técnicas de filtrado. Estas técnicas permiten resaltar o suprimir, de forma selectiva, información contenida en una imagen para destacar u ocultar algunos elementos de ella. Con la aparición de las últimas familias de FPGAs a principios del año 2000 y hasta el día de hoy, se han incorporado nuevos recursos internos orientados hacia el procesamiento de imágenes. Esto ha provocado que, dentro de esta área, se haya producido un incremento en el desarrollo de algoritmos de visión de alto nivel.

La aparición de nuevos lenguajes de programación para FPGA con un nivel de abstracción mayor que VHDL o Verilog, como pueden ser *Bach-C*, *Impulse-C*, *System-C* y sobre todo *Handel-C*, han permitido trasladar algoritmos codificados en lenguajes de alto nivel para PC a hardwares reconfigurables [8]. En la mayoría de los trabajos desarrollados hasta la fecha relacionados con este tipo de lenguaje, la optimización del sistema se basa en la partición del algoritmo en elementos que puedan ejecutarse en paralelo o en el diseño de arquitecturas donde los elementos de procesado pueden trabajar en paralelo (como por ejemplo son las GPUs [4]).



Fig. 2.3: Cámara digital.

Actualmente, para poder procesar y entender una imagen o secuencia de imágenes, la visión artificial tiene como elemento fundamental las cámaras digitales, las cuales tienen la función de captar la luminiscencia de cada punto en una escena, tal y como lo hace el ojo humano. Esta cámara transforma los niveles de voltaje que recibe en una señal digital. Todo este proceso incluye errores tanto en la percepción como en la transformación analógico-digital, errores que se acumulan formando ruido en la imagen. De aquí la necesidad de aplicar un tratamiento a esta organización de datos dentro de la matriz de la imagen, para interpolar el valor de cada píxel con ruido a uno más próximo al real de forma matemática, lo que se denomina procesamiento digital de imágenes. Este procesamiento se lleva a cabo mediante diferentes técnicas, como lo son el filtrado espacial y el filtrado puntual [16].

La reducción de ruido en procesamiento de imágenes ha tomado gran relevancia en áreas como la medicina, ya que este ruido en las imágenes puede ocasionar un diagnóstico erróneo

[2, 3]. Asimismo, este procesamiento también puede ayudar a las máquinas a entender posiciones, velocidad y aceleración, en aplicaciones como robótica y procesos industriales, mediante la detección de bordes. Este tipo de aplicaciones requieren un procesamiento basado en restricciones de tiempo (tiempo real) por lo que los investigadores han evaluado distintas plataformas que existen para implementar algoritmos de procesamiento digital de imágenes, donde las FPGAs han destacado de forma significativa, gracias al alto grado de procesamiento en paralelo y pipeline, además de trabajar a una alta frecuencia de operación.

El filtrado de imágenes es una parte fundamental en los sistemas de clasificación de imágenes mediante Deep Learning y Visión Artificial. En estos procesos pueden distinguirse 5 etapas generales: adquisición, preprocesado, segmentación, representación/descripción y reconocimiento/interpretación. Tanto en las etapas de preprocesado (con técnicas como la eliminación de ruido o el realce de detalles) como en la segmentación (detección de bordes o de puntos aislados) e incluso en la representación, existen diferentes técnicas y métodos de filtrado de imagen con diferentes finalidades pero con un nexo común: la computación sobre los píxeles de las imágenes.

Capítulo 3

Materiales y fundamentos teóricos

En este Capítulo se pretende realizar una explicación detallada de los principales materiales utilizados y de los fundamentos teóricos sobre los que se basa el proyecto creado. En primer lugar, se realiza una introducción a la placa que se utiliza en el proyecto y se dan algunas características importantes sobre la FPGA que posee, comentando algunas de sus ventajas. Posteriormente, se realiza un estudio sobre las distintas interfaces que se van a utilizar y así entender el diseño que se realiza más adelante en el Capítulo 4. De la misma forma, se estudia en profundidad el sensor de imagen utilizado y las características que presenta. Finalmente se realiza un análisis de los fundamentos teóricos que residen en el filtrado de imagen y se hace un estudio de las diferentes técnicas que se emplean.

3.1. Placa de desarrollo *Nexys 4 DDR*

La placa de desarrollo Nexys 4 DDR es una plataforma de desarrollo de circuitos digitales completa y lista para usarse, diseñada por el fabricante Digilent y basada en la FPGA Artix-7 de Xilinx [17]. Esta placa es capaz de alojar grandes circuitos digitales y potentes procesadores embebidos.

Asimismo, posee una serie de periféricos y módulos externos que sirven de mucha utilidad para evaluar y probar las distintas aplicaciones de la FPGA; de los cuales algunos se utilizan para la realización de este proyecto. Algunos de estos periféricos son, por ejemplo, un acelerómetro, un sensor de temperatura, un micrófono digital MEMS, un amplificador de altavoces y varios dispositivos de E/S como interruptores, botones, LEDs, etc (Tabla 3.1).

En Fig. 3.1 se pueden observar todos los periféricos que se encuentran en esta tarjeta de desarrollo. La elección este modelo para llevar a cabo este proyecto se debe, en parte, a la facilidad de acceso a él, debido a su disponibilidad en el laboratorio de la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad Politécnica de Madrid. Por otro lado, debido a la potencia de procesamiento que esta posee y a la facilidad de conectarle otros dispositivos, hacen de ella una excelente herramienta de trabajo y estudio de la tecnología.

La Nexys 4 DDR es compatible con Vivado Design Suite de Xilinx, así como con el conjunto de herramientas ISE, que incluye ChipScope y EDK (Embedded Development Kit). Xilinx ofrece versiones gratuitas de estas herramientas en WebPACK, por lo que los diseños pueden ser implementados sin costo adicional. En comparación con su versión anterior, la Nexys 4, sustituye la 16 MiB CellularRAM por una memoria SDRAM DDR2 de 128 MiB.

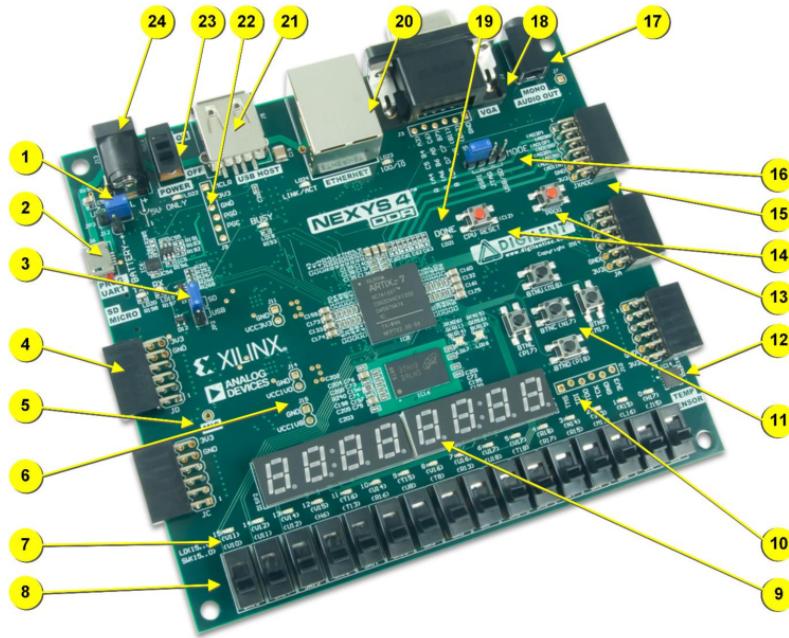


Fig. 3.1: Placa de desarrollo Nexys 4 DDR [17].

La Tabla 3.1 representa los componentes que ofrece la placa de desarrollo utilizada para este proyecto, los cuales han sido enumerados en función de Fig. 3.1. Además de los que se indican en la tabla, la Nexys 4 DDR incluye un oscilador de cristal de 100 MHz para generar la frecuencia de reloj máxima a la que puede trabajar la FPGA.

Tabla 3.1: Componentes que contiene la placa de desarrollo Nexys 4 DDR.

Ref	Descripción del componente	Ref	Descripción del componente
1	Puente de selección de potencia y cabezal de batería	8	Interruptores deslizantes
2	Puerto USB compartido: UART y JTAG	9	Ocho displays de 7 segmentos
3	Puente de configuración externa (SD o USB)	10	Puerto JTAG para cable externo (opcional)
4	Puerto(s) Pmod	11	Cinco pulsadores
5	Micrófono	12	Sensor de temperatura
6	Punto(s) de prueba de la fuente de alimentación	13	Botón de reset de la configuración de la FPGA
7	Matriz de LEDs (16)	14	Botón de reset de la CPU (para soft cores)

Ref	Descripción del componente	Ref	Descripción del componente
15	Puerto Pmod de señal analógica (XADC)	20	Conector Ethernet
16	Puente de modo de programación	21	Conector host USB
17	Conector de audio	22	Puerto de programación PIC24 (uso en fábrica)
18	Conector VGA	23	Interruptor de encendido
19	LED indicador de programación realizada en la FPGA	24	Enchufe de alimentación

3.1.1. FPGA Artix-7

La revolución digital ha determinado un cambio en el diseño de FPGAs, ya que la competencia que existe en mercados como el aeroespacial, la defensa, la medicina, la industria y la electrónica de consumo exigen un elevado rendimiento en un amplio rango de trabajo. Sin sacrificar el rendimiento, los desarrolladores deben ser capaces de mejorar los modelos para obtener un mayor ancho de banda de procesamiento, así como ampliar el alcance de las aplicaciones mientras se mantiene la batería (recurso critico) al mínimo.

La familia Artix-7 es capaz de reducir el consumo de energía a la mitad con respecto a la anterior generación, a la vez que proporciona los mejores transceptores de su clase y capacidades de procesamiento de señal para aplicaciones de alto ancho de banda. Estas FPGAs están construidas sobre el proceso HPL (High Pressure Lamine) de 28 nm, por lo que ofrecen el mejor rendimiento por vatio de su clase. Estos dispositivos son ideales para productos como equipos médicos portátiles, radios militares y equipos inalámbricos de infraestructura compacta. Eso es debido a que satisfacen las necesidades actuales del mercado relacionadas con el tamaño, peso, consumo y costes. En comparación con la generación anterior, Artix-7 ofrece las siguientes características [18]:

- Poseen el doble de lógica, 2.5 veces más de RAM, y 5.7 veces más de *slices* de DSPs que las *Spartan®-6* (Familia de Xilinx inferior).
- Estándares de E/S con velocidades de hasta 1,25 Gb/s
- Transceptores de 6,6 Gb/s que permiten un ancho de banda pico de 211 Gb/s (*full duplex*).
- Procesamiento de producción probado de 28 nm.
- Interfaz de memoria integrada para un acceso optimizado.
- Bloques IP integrados para reducir el tiempo de desarrollo y el riesgo.
- Asistentes integrados para un rápido desarrollo de los sistemas de bloques.
- Rendimiento de subvatos que varía de 13 mil a 2 millones de celdas lógicas.

Las características anteriormente mencionadas determinan esta familia como el complemento perfecto para aplicaciones en las que el coste es un factor importante, ofreciendo propiedades de gamas altas en casi todas las características de rendimiento. En concreto, la ventaja más significativa que ofrecen los dispositivos Artix-7 es que poseen los transceptores más optimizados de la industria, así como el mayor rendimiento y el menor consumo. Además, la lógica y señales de procesamiento, la memoria integrada, E/S LVDS (*Low-Voltage Differential Signaling*), junto a las interfaces de memoria permite desarrollar aplicaciones específicas que satisfagan las necesidades del momento.

Xilinx ofrece tanto el Artix-7 FPGA AC701 kit como el Artix-7 50T FPGA Evaluation Kit, que permiten la creación rápida de prototipos para aplicaciones sensibles a costes. Estos kits incluyen todos los componentes básicos de hardware, herramientas de diseño, IP y diseños de referencia pre-verificados para empezar con la familia Artix-7.

Transceiver Optimization at the Lowest Cost and Highest DSP Bandwidth (1.0V, 0.95V, 0.9V)									
	Part Number	XC7A12T	XC7A15T	XC7A25T	XC7A35T	XC7A50T	XC7A75T	XC7A100T	XC7A200T
Logic Resources	Logic Cells	12,800	16,640	23,360	33,280	52,160	75,520	101,440	215,360
	Slices	2,000	2,600	3,650	5,200	8,150	11,800	15,850	33,650
	CLB Flip-Flops	16,000	20,800	29,200	41,600	65,200	94,400	126,800	269,200
Memory Resources	Maximum Distributed RAM (Kb)	171	200	313	400	600	892	1,188	2,888
	Block RAM/FIFO w/ ECC (36 Kb each)	20	25	45	50	75	105	135	365
	Total Block RAM (Kb)	720	900	1,620	1,800	2,700	3,780	4,860	13,140
Clock Resources	CMTs (1 MMCM + 1 PLL)	3	5	3	5	5	6	6	10
I/O Resources	Maximum Single-Ended I/O	150	250	150	250	250	300	300	500
	Maximum Differential I/O Pairs	72	120	72	120	120	144	144	240
Embedded Hard IP Resources	DSP Slices	40	45	80	90	120	180	240	740
	PCIe® Gen2 ⁽¹⁾	1	1	1	1	1	1	1	1
	Analog Mixed Signal (AMS) / XADC	1	1	1	1	1	1	1	1
	Configuration AES / HMAC Blocks	1	1	1	1	1	1	1	1
Speed Grades	GTP Transceivers (6.6 Gb/s Max Rate) ⁽²⁾	2	4	4	4	4	8	8	16
	Commercial Temp (C)	-1, -2	-1, -2	-1, -2	-1, -2	-1, -2	-1, -2	-1, -2	-1, -2
	Extended Temp (E)	-2L, -3							
	Industrial Temp (I)	-1, -2, -1L							

Fig. 3.2: Tabla comparativa de los modelos de la familia Artix-7.

En este trabajo se utiliza uno de los dispositivos que ofrece la familia Artix-7, el modelo XC7A100T, cuyas características, en comparación con las de otros modelos, se pueden observar en la tabla de Fig. 3.2.

Este dispositivo contiene unas 100 mil celdas lógicas para trabajar y un total de 4860 kb = 607,5 kB de BRAM, además de 1188 kb = 148,5 kB de *Distributed RAM*.

La estructura básica de las FPGA actuales de Xilinx se compone de los siguientes elementos:

- **Configurable Logic Blocks (CLB)** [19]: principales recursos lógicos para implementar circuitos secuenciales y combinacionales. En su interior figuran dos slices, los cuales están formados por:
 - **Look-Up Table (LUT)**: elemento encargado de realizar operaciones lógicas. Cada slice posee 4 LUTs de 6 entradas.
 - **Flip-Flop (FF)**: registro que almacena el resultado obtenido de la LUT. Cada LUT posee 2 FF a la salida. Pueden configurarse como latches.

– Multiplexores y lógica de acarreo.

- **Block RAM (BRAM)** [20]: pueden configurarse como memorias de un puerto o de doble puerto (accesos independientes).
- **Digital Signal Processing (DSP48) slices** [21]: bloques dedicados de DSP, personalizables y de bajo consumo, que combinan altas velocidades con un tamaño pequeño (manteniendo la flexibilidad).
- **Wires (Cables)**: permiten la interconexión de unos elementos con otros.
- **Pads de E/S**: puertos disponibles físicamente que transportan datos hacia dentro y hacia fuera de la FPGA.

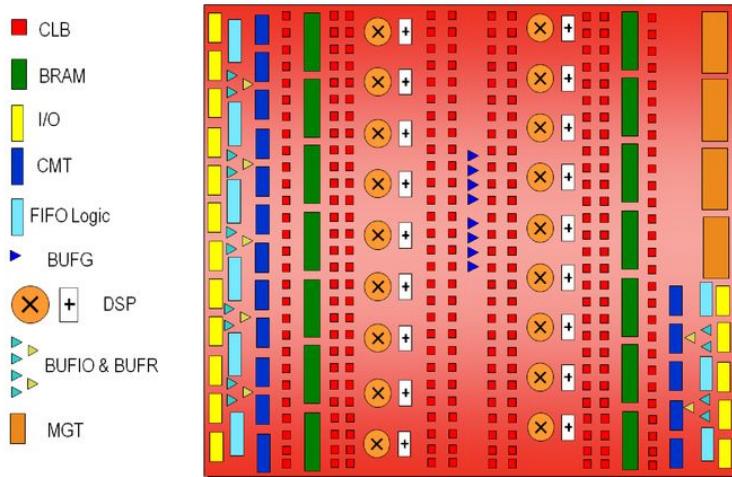


Fig. 3.3: Arquitectura de la serie Artix-7 [14].

Sin embargo, existen una serie de bloques computacionales y de almacenamiento de datos adicionales que incrementan la densidad computacional y eficiencia del dispositivo. Estos elementos son, por ejemplo, memorias embebidas para almacenar datos de forma distribuida, PLLs (Phase-locked loops) para manejar la estructura interna de la FPGA a distintas frecuencias de reloj, transmisores-receptores serie de alta velocidad o controladores de memoria “off-chip”. La combinación de estos elementos da lugar a una arquitectura con flexibilidad de implementar cualquier algoritmo software ejecutable en un procesador.

En particular, la serie Artix-7 tiene la arquitectura que se representa en Fig. 3.3. Esta arquitectura responde a la cuarta generación de Arquitectura ASMBL (del inglés *Application Specific Modular Block Architecture*) [6, 12, 13]. Básicamente consiste en una disposición de los recursos en el interior de la FPGA diferente a la tradicional, concretamente en columnas de bloques. Esta disposición permite que el tamaño de las celdas sea independiente del escalado de características, es decir, que el aumento de recursos no requiere un aumento directo del tamaño del array. Esto permite una unificación de la arquitectura entre las diferentes familias aunque estas posean diferentes relaciones de recursos.



Fig. 3.4: Layout de la Serie-7 de FPGAs de Xilinx [12].

Como se puede apreciar en Fig. 3.4 todos los dispositivos tienen dos columnas de E/S paralelas, existiendo dos tipos diferentes: alto rango (soporta estándares de hasta 3,3 V) y alto rendimiento (soporta estándares de hasta 1,8 V). Disponen de desplazador de fase, IO FIFO y IO PLL.

Junto con estas columnas se encuentran las columnas de control de reloj (CMT - *Clock Management Tile*) que permite interfaces de E/S de mayor velocidad, con recursos dedicados. Además, el ruteado de reloj circula por la columna central y se reparte por el resto de la FPGA. Cada región de reloj tiene una altura de 50 CLBs, 25 filas por encima y 25 por debajo. Se reducen los sesgos en la señal de reloj (*clock skew*) gracias a estas distribuciones.

3.1.2. Interfaz *Pmod*

Como ya se introdujo en el Capítulo 1, para la comunicación entre el sensor de imagen y la FPGA se utiliza esta interfaz que ofrece la Nexys 4 DDR [5].

Esta interfaz se utiliza para conectar módulos periféricos de baja frecuencia y bajo número de pines de E/S a tarjetas controladoras del host. Hay versiones de seis y doce pines, en este caso se utiliza la versión de doce, la cual posee dos interfaces de seis pines apiladas. Como se aprecia en Fig. 3.5, son ocho pines de señal de E/S, dos pines de alimentación y dos pines de masa. Para la comunicación con el sensor la interfaz permite una configuración en I^2C , pudiendo conectarse los pines de la tarjeta del sensor directamente a los pines de la Nexys 4 DDR.

Las especificaciones eléctricas establecen una fuente de alimentación lógica de 3,3 V y que las señales se ajusten a las convenciones lógicas *LVCMOS 3,3 V* o *LVTTL 3,3 V*. Los pines de E/S de las FPGAs Xilinx poseen una capacidad simétrica de 24 mA, porque son

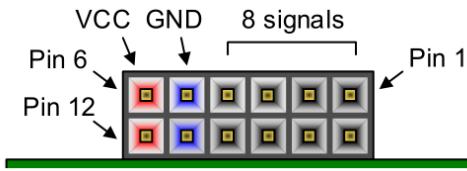


Fig. 3.5: Pines de la interfaz Pmod.

dirigidos directamente desde la FPGA. Estos cuentan también con diodos de protección ESD y resistencias de $200\ \Omega$ para limitar las corrientes de cortocircuito involuntario de las clavijas y para proteger frente conflictos de entrada y salida.

A parte del protocolo I^2C , esta interfaz se puede configurar, también, en base a cinco tipos diferentes de protocolos de comunicación: GPIO, SPI o SPI expandido, UART o UART expandida, puente H y doble puente H.

3.1.3. Interfaz VGA

Inicialmente, VGA (del inglés *Video Graphics Array*) se utiliza para denominar a la tarjeta gráfica que comercializó IBM® por primera vez en 1988. Actualmente, se utiliza para referirse tanto al estándar de video de pantallas analógicas, como al conector DE-15 o a la resolución de pantalla 640x480.

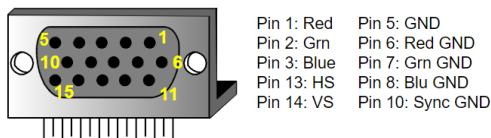


Fig. 3.6: Pines de conexión de la interfaz VGA.

Este conector consta de 15 pines distribuidos en tres filas, el nombre de cada uno se muestra en Fig. 3.6. En la Nexys 4 DDR se utilizan 14 señales que produce la FPGA para crear esta comunicación, concretamente son 4 bits por color (utilizando los pines 1, 2 y 3) y 2 señales de sincronización (con los pines 13 y 14); estas últimas proporcionan una referencia de posición sobre la pantalla.

Como el interfaz VGA trabaja con señales analógicas para las señales de color, es necesario la conversión de la señal digital. Para ello, se utilizan divisores resistivos, uno por cada bit de las tres señales de color de la FPGA. Puesto que se tienen 4 bits, se consiguen crear 16 niveles de señal en cada color. De esta forma se permite tener, finalmente, una señal RGB de 12 bits que puede crear 4096 colores diferentes.

3.1.3.1. Funcionamiento de un monitor de rayos catódicos

Para poder configurar correctamente los controladores de esta interfaz es necesario conocer la arquitectura y el funcionamiento del elemento destino. Las pantallas de tubos de rayos catódicos o CRT⁶ utilizan tres haces de electrones (uno para cada color RGB) para energizar el fósforo del lado interior de la pantalla, en Fig. 3.7 se puede observar una representación de cómo se disponen en la pantalla los distintos elementos. Estas son predecesoras de las pantallas LCD pero poseen los mismos tiempos en las señales de sincronización.

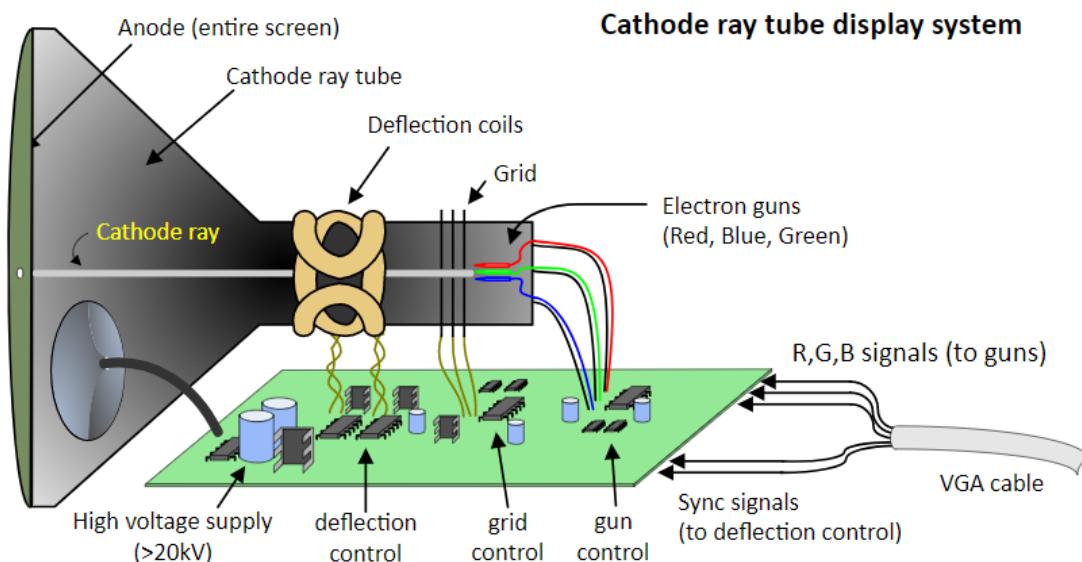


Fig. 3.7: Pantalla CRT de color.

Los “cañones de electrones” (*electron guns*) son cátodos, orientados internamente hacia la pantalla, que se calientan y se sitúan muy próximos a una placa anular positivamente cargada llamada “red” (*grid*). Inicialmente, es la red la encargada de acelerar las partículas que llegan a los cátodos, pero posteriormente es la misma pantalla la que atrae estas partículas, pues se carga a una tensión muy elevada (20 kV o más). El brillo sobre la pantalla depende de la intensidad que posean los electrones que se lanzan desde los cañones.

Como se puede ver en Fig. 3.7, entre la red y la pantalla se ubican unas bobinas de cobre que producen unos campos electromagnéticos ortogonales a la dirección de los haces. Se encargan de desviar la dirección del haz para incidir en los distintos puntos de la pantalla, siguiendo una trayectoria horizontal y vertical como la que se observa en Fig. 3.8 (de izquierda a derecha y de arriba a abajo). La información solo se muestra en la pantalla cuando el haz sigue la trayectoria indicada anteriormente, pero este haz debe recolocarse al terminar cada fila para apuntar al comienzo de la siguiente y al terminar todas las filas apuntando hacia el comienzo de la primera fila. Este reajuste es el tiempo que debe tenerse en cuenta para la configuración del controlador.

⁶Tubo de rayos catódicos (del inglés *Cathode Ray Tube*), es una tecnología para la visualización de imágenes mediante haces de electrones de amplitud modulada sobre una pantalla de vidrio recubierta de fósforo y plomo.

De este modo, se puede ajustar la resolución de la pantalla controlando el tamaño de los haces, la frecuencia con la que se puede trazar el haz a través de la pantalla y la frecuencia con la que se puede modular el haz de electrones.

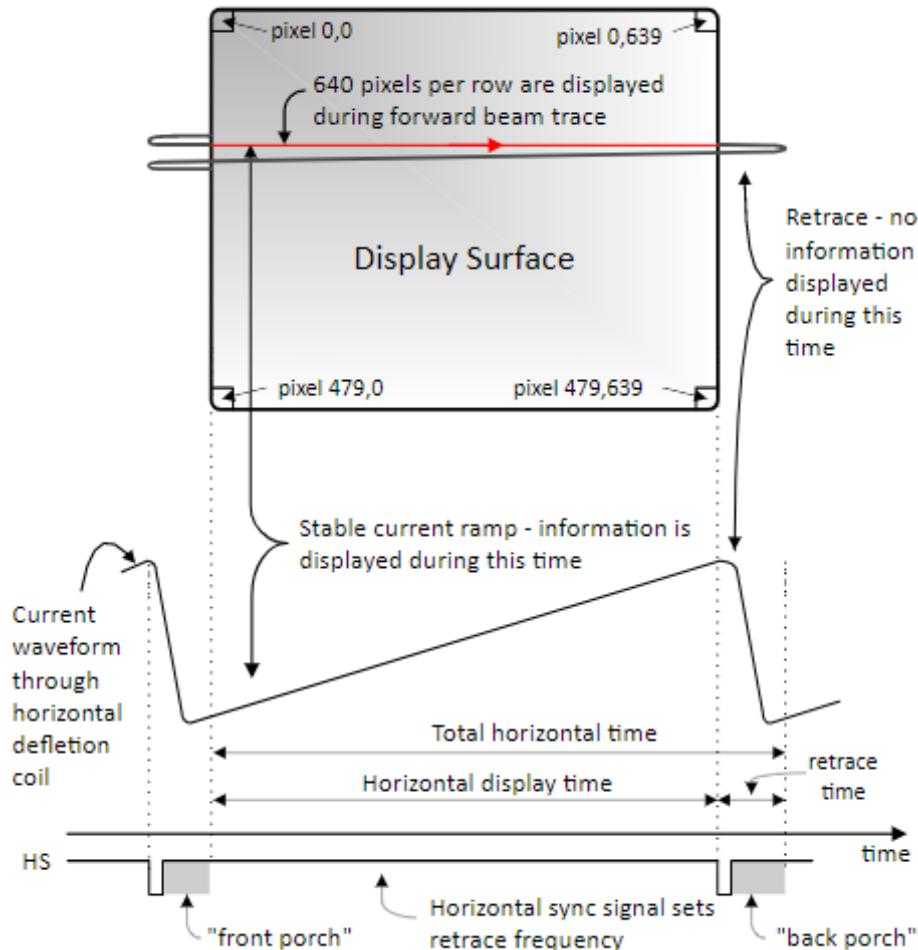


Fig. 3.8: Sincronización horizontal del haz de electrones

3.2. Sensor de imagen CMOS

El método de obtención de imágenes es una parte muy importante en este proyecto. Es crucial poseer un adecuado método de captura de imágenes con la finalidad de conseguir un buen sistema de filtrado de imágenes que pueda ser aplicado sobre sistemas con requisitos de tiempo real. Para poder entender más adelante cómo se diseña y crea el sistema de captura de imágenes, es necesario conocer cómo funciona el sensor que se va a utilizar para ello. En esta sección se hace una breve explicación de la tecnología que se va a utilizar y así más adelante se entenderá el proceso de diseño.

Un sensor de imagen consiste en un conjunto de píxeles, en el que cada uno de los cuales contiene un fotodetector (en el fondo una unión PN) que convierte la luz incidente en photocorriente. Como se puede observar en Fig. 3.9, las señales de tensión de carga producidas por cada fotoreceptor se leen en una fila cada vez, al igual que se produce en una memoria RAM, utilizando circuitos de selección de fila y columna [22, 23].

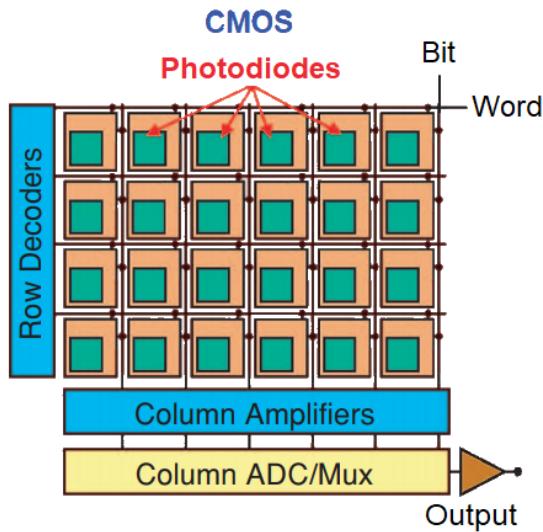


Fig. 3.9: Arquitectura interna de un sensor de imagen CMOS.

También posee algunos de los circuitos de lectura necesarios para convertir la photocorriente en carga eléctrica o tensión y para leerla fuera del array de imagen. El porcentaje de área ocupada por el fotodetector en un píxel se conoce como factor de relleno. El resto de los circuitos de lectura están situados en la periferia del array y son multiplexados por los píxeles. Los tamaños de los arrays pueden ser tan grandes como decenas de megapíxeles para aplicaciones de gama alta, mientras que los tamaños de píxeles individuales pueden ser tan pequeños como $2 \times 2 \mu\text{m}$. Una matriz de microlentes se deposita típicamente encima de la matriz de píxeles para aumentar la cantidad de luz que incide en cada fotodetector [23].

3.2.1. OV7670 CameraChip™

El sensor de imagen que se emplea en este proyecto es el sensor OV7670 CAMERACHIP™ con tecnología OmniPixel®, desarrollada por la compañía OmniVision® Technologies Inc. [24, 25].

A parte de comportarse como sensor de imagen CMOS de bajo voltaje, este dispositivo es un SoC y tiene una amplia capacidad de procesado de imagen en un solo chip. Es capaz de ofrecer distintas funcionalidades de control de video con una matriz de imagen que puede llegar a funcionar a una velocidad de 30 fps (frames por segundo) en resolución VGA. La programación de este control de imagen se puede realizar a través de la interfaz SCCB (*Serial Camera Control Bus*), la cual es compatible con interfaz I^2C , lo que permite la posibilidad

de crear un sistema de programación de los registros con la FPGA siguiendo ese protocolo de comunicación.

Algunas de las funcionalidades de procesamiento de imagen son: control de exposición automática (AEC), control del balance de blancos (AWB), control automático de ganancia (AGC), cancelación de ruido, control del brillo (automático o manual), de la saturación, gamma y otros parámetros. Además, es capaz de trabajar a distintas resoluciones (VGA, QVGA, CIF y QCIF) y con diferentes espacios de color (RGB444, RGB555, RGB565, YUV). En Fig. 3.10 se puede entender de forma más detallada cómo se organiza el chip.

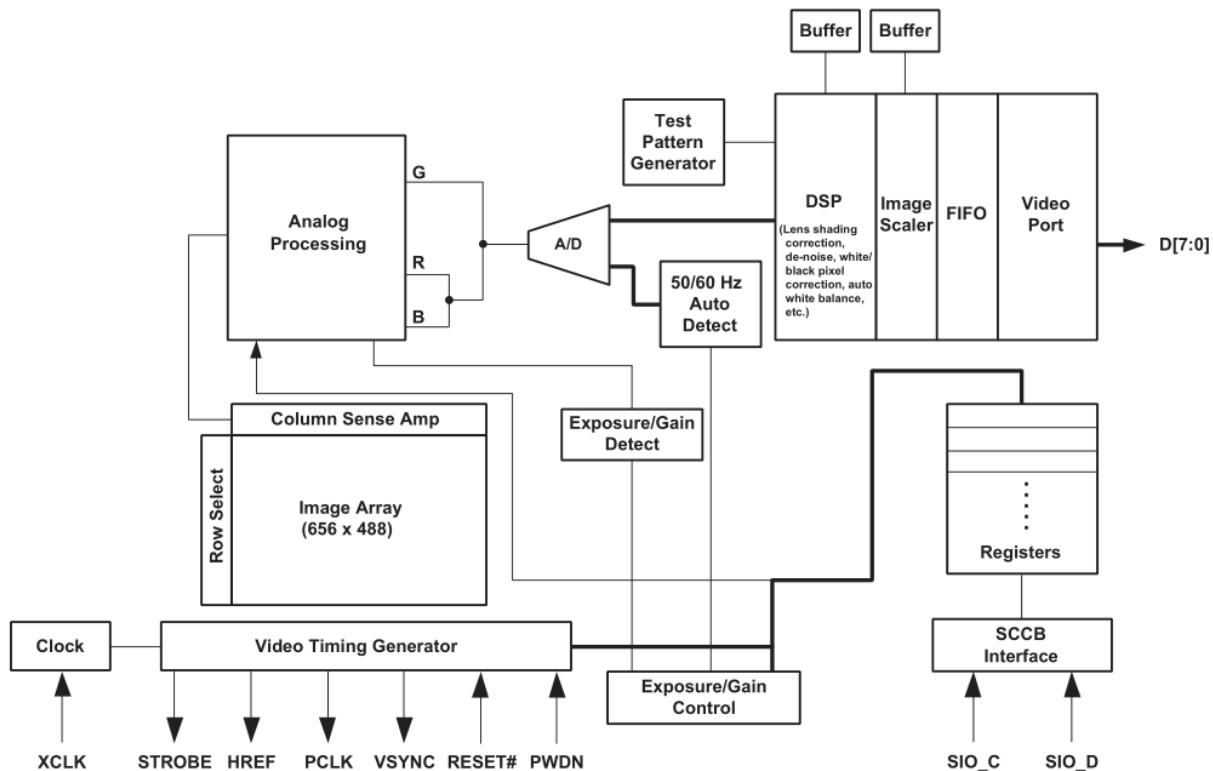


Fig. 3.10: Diagrama de bloques del SoC OV7670.

De entre todos los módulos que se pueden observar en la imagen anterior, los más importantes son:

- El **array de captura** de imagen dispone de 656x488 píxeles, de los cuales solo están activos 640x480, permitiendo capturar la imagen en resolución VGA.
- El **generador de tiempos** controla el sensor de imagen y la generación de fotogramas, la generación y distribución de las señales internas, la generación de las señales de sincronización, la velocidad de fps y, además, incluye el control de exposición automático.
- El **procesador analógico** se encarga de las funcionalidades de procesamiento que se han comentado anteriormente.
- El **conversor A/D** tiene una precisión de 10 bits y toma la salida del procesador

analógico a un máximo de 12 MHz, es totalmente síncrono con la velocidad de captura de los píxeles.

- El **procesador digital de señal (DSP)** controla la interpolación de la señal original a RGB e incorpora un control de calidad de imagen con una serie de funcionalidades adicionales.

- Mediante la **interfaz SCCB** se accede a los valores de los registros internos para controlar el modo de operación del sensor de imagen.

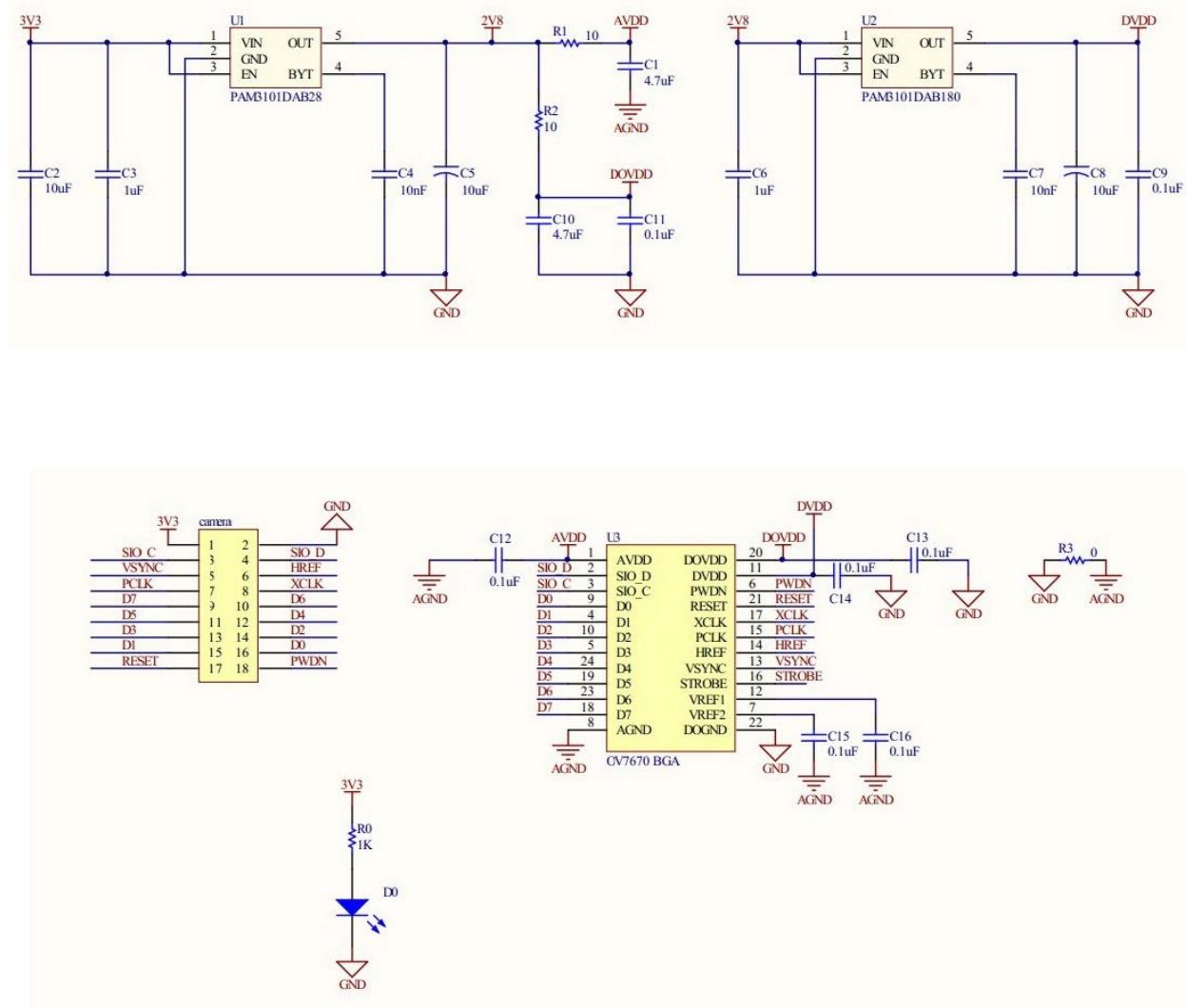


Fig. 3.11: Esquemático de la placa OV7670 que se ha adquirido para el proyecto.

Para trabajar con este sensor se ha adquirido una placa que lo integra, Fig. 3.12. En ella se dispone de dos reguladores de tensión, de condensadores para absorber los posibles ruidos en las tensiones de alimentación y de un conjunto de resistencias para los divisores de tensión y otros pequeños ajustes. De esta forma, se puede alimentar la placa a 3,3 V e internamente se divide este valor para obtener el resto de tensiones que el SoC necesita para alimentar los distintos módulos. En los circuitos de la parte alta de Fig. 3.11 se pueden ver estos dos

reguladores, el primero (a la izquierda) baja de 3,3V a 2,8V para la alimentación de los pines AVDD y DOVDD. El segundo circuito (a la derecha) toma la reducción creada para alimentar el pin DVDD a 1,8V [24].



Fig. 3.12: Fotografía de la placa OV7670 que se utiliza en este proyecto.

Los pines de esta placa, que van conectadas a los JTAG de la Nexys 4 DDR (apartado 3.1.2), se observan en el bloque denominado “camera” del esquemático que se describe en Fig. 3.11. Para realizar una correcta implementación del sistema de captura de imagen se van a describir estas señales de entrada y salida de la placa OV7670.

- **3V3:** pin de alimentación a 3,3 V.
 - **GND:** pin de masa.
 - **PWDN:** IN - Selector para apagar el sensor (0: modo normal, 1: apagado).
 - **RESET:** IN - Limpia los registros a los valores por defecto (activo a nivel bajo).
 - **XCLK:** IN - Reloj del sistema.
 - **SIOD:** IN/OUT - Datos de la interfaz serie SCCB.
 - **SIOC:** IN - Reloj de la interfaz serie SCCB.
 - **DATA[7:0]:** OUT - Bits de salida para los datos de video.
 - **HREF:** OUT - Referencia horizontal, indica el fin de una fila de píxeles cuando se pone a nivel bajo (ver Fig. 3.13 y Fig. 3.14).
 - **VSYNC:** OUT - Sincronización vertical, indica el fin de un fotograma cuando se pone a nivel alto (ver Fig. 3.14).
 - **PCLK:** IN - Reloj de píxel, indica cuándo se puede leer un dato en los pines de salida (flancos ascendentes).

Como se puede observar en Fig. 3.13, la señal PCLK generada por el generador de temporización de video marca el ritmo al que se capturan los píxeles y establece, en los flancos de subida de la señal, la correcta disponibilidad de los datos en la salida del sensor. De esta forma mientras se transmiten todos los datos de una misma fila HREF se mantiene a nivel alto.

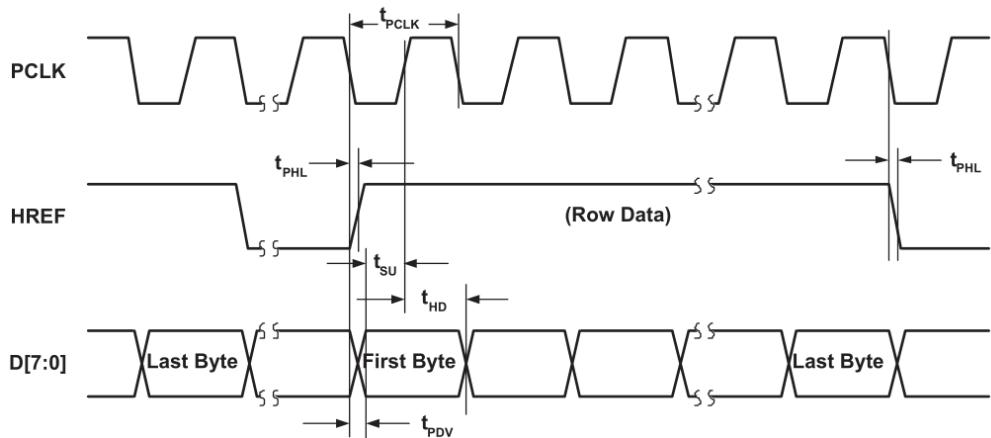


Fig. 3.13: Temporización horizontal de los datos de salida.

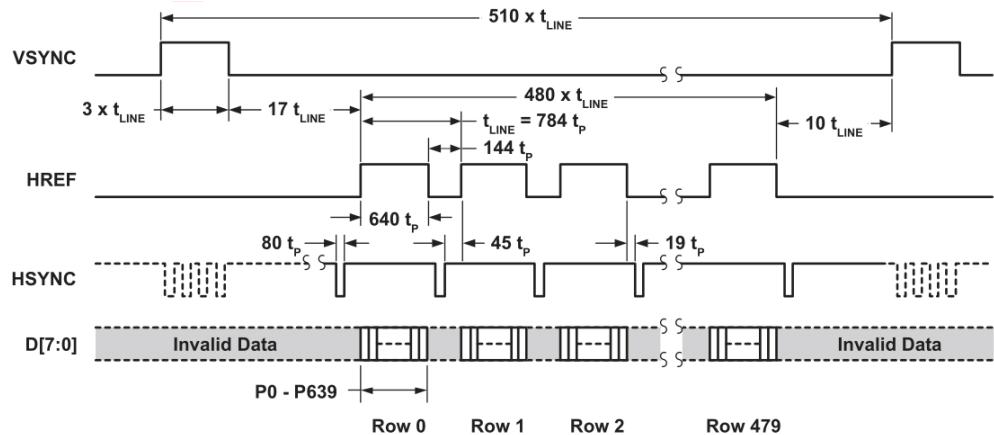


Fig. 3.14: Temporización de los datos de cada fotograma.

En Fig. 3.14, sobre una escala de tiempo de mayor tamaño, se observan los flancos de subida y bajada de la señal de HREF, que indican cada una de las filas que se van transmitiendo y que, mientras todas las filas pertenezcan a la misma imagen, VSYNC se mantiene a nivel bajo.

Finalmente, la última consideración a tener en cuenta antes de realizar el diseño del sistema de captura de imágenes es que para cada formato de color los datos de los píxeles tienen una organización diferente. En Tabla 3.2, Tabla 3.3 y Tabla 3.4 se detalla de forma clara en qué posiciones de los dos primeros Bytes transmitidos se ubica cada bit. El protocolo

de comunicación está configurado de forma que se repite el mismo patrón cada dos Bytes, es decir, las tablas representan la organización del mensaje de los datos de cada píxel.

Por ejemplo, en Tabla 3.2 se observa que hay 5 bits para el color rojo (R), 6 bits para el color verde(G) y 5 bits para el color azul(B) y puesto que en total suman 16 bits, estos quedan repartidos de forma exacta en 2 Bytes. Como resultado se tiene que cada píxel necesita dos Bytes para transmitirse. La diferencia en los otros dos formatos es que se reduce el número de bits de cada señal de color y por lo tanto sobran huecos en el paquete de bits.

Tabla 3.2: Formato RGB565.

Primer Byte							
D[7]	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]
R_4	R_3	R_2	R_1	R_0	G_5	G_4	G_3

Segundo Byte							
D[7]	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]
G_2	G_1	G_0	B_4	B_3	B_2	B_1	B_0

Tabla 3.3: Formato RGB555.

Primer Byte							
D[7]	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]
X	R_4	R_3	R_2	R_1	R_0	G_4	G_3

Segundo Byte							
D[7]	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]
G_2	G_1	G_0	B_4	B_3	B_2	B_1	B_0

Tabla 3.4: Formato RGB444.

Primer Byte							
D[7]	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]
X	X	X	X	R_3	R_2	R_1	R_0

Segundo Byte							
D[7]	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]
G_3	G_2	G_1	G_0	B_3	B_2	B_1	B_0

3.3. Filtrado de imagen

Este proyecto tiene por finalidad la realización de un proceso de filtrado en hardware a una serie de fotografías que se van capturando. Por ello, con la intención de contextualizar esa parte del sistema se realiza un estudio de diferentes técnicas y procesos de filtrado.

En primer lugar, se entiende que una imagen puede interpretarse como una función bidimensional $z = F(x, y)$ donde x e y son coordenadas espaciales, y z es el valor de la intensidad de la imagen en el punto (x, y) . Las imágenes analógicas son siempre funciones continuas y, por ello, para convertir una imagen a formato digital es necesario hacer un proceso de discretización tanto para las coordenadas como para la intensidad. La digitalización de las coordenadas se llama muestreo, mientras que la digitalización de la intensidad se denomina cuantización [26].

La imagen digital en escala de grises viene dada a partir de la matriz $z_{ij} = F(x_i, y_j)$. Cada entrada de la matriz es un píxel y el número de píxeles (el orden de la matriz) define la resolución de la imagen digital. El valor de cada píxel está cuantificado en un número determinado de bits de 0 a $(2^p - 1)$ posibles. Las variaciones cambian según el contenido en bits de una imagen (calidad de imagen). Habitualmente se trabaja con $p = 8$, por lo que el rango de intensidades varía de un valor de 0 (negro) a 255 (blanco). Cuando se trata de una imagen digital en color se trabaja con tres matrices: R, G y B, que proporcionan las intensidades correspondientes a los colores primarios rojo, verde y azul. En este proyecto se emplean 4 bits por cada color (apartado 3.1.3), lo que provoca que se obtiene una gama de 4096 colores.

El filtrado de imágenes consiste en la modificación de las matrices correspondientes a la imagen digitalizada mediante el procedimiento que se halla diseñado. Los algoritmos de filtrado pueden clasificarse en lineales o no-lineales, de esta forma, los filtros basados en operaciones puntuales suelen ser lineales, mientras que los filtros basados en regiones del espacio o en los vecinos son no-lineales. La diferencia principal es que los operadores en los filtros lineales son invertibles, es decir, la operación puede deshacerse aplicando el operador inverso [10, 27].

Los filtros espaciales modifican la contribución de determinados rangos de frecuencias espaciales en una imagen. El término espacial hace referencia a la aplicación directa del filtro sobre la imagen y no sobre su transformada, a pesar de hacer referencia a las frecuencias espaciales. Sin embargo, existe una relación entre los filtros espaciales y el filtrado en el dominio de frecuencias. De esta forma se dice que una imagen está formada por componentes de frecuencia que varían de bajas a altas frecuencias [28].

- Existen **bajas frecuencias** en una imagen cuando las transiciones de brillo apenas cambian entre los píxeles de la imagen. Los filtros de este tipo actúan del mismo modo, suavizan (*smoothing*) las imágenes produciendo una cierta difuminación o pérdida de nitidez. Se suelen utilizar en la reducción del ruido o en la difusión de bordes.
- Las **altas frecuencias** están asociadas a transiciones rápidas de brillo, es decir, cuando hay bordes abruptos (*sharpening*) entre regiones adyacentes existe alta frecuencia. Los filtros

que se consideran de alta frecuencia se utilizan con el efecto contrario, para resaltar los detalles finos o para recuperar detalles perdidos en una mala adquisición de la imagen.

3.3.1. Filtros de convolución

Uno de los procedimientos más utilizados en el procesamiento de señales y de imágenes es el de la convolución de matrices. Cuando se aplica sobre imágenes, en cada píxel resultante se tiene en cuenta la influencia de los píxeles vecinos, por ello responde a un tipo de filtrado espacial.

Antes de trasladar el concepto al ámbito del procesamiento de imagen, es necesario destacar que una convolución no es más que un operador matemático que transforma dos funciones f y g en una tercera función h , la cual representa la magnitud en la que se superponen f y una versión trasladada e invertida de g . Se trata de un tipo general de media móvil.

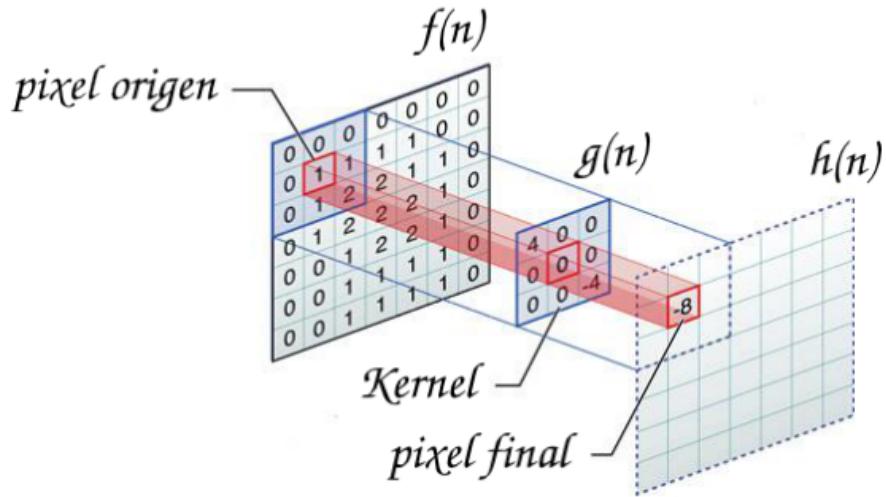


Fig. 3.15: Proceso de convolución [28].

Definición. La convolución de f y g se denota $(f * g)$ y se define como la integral del producto de ambas funciones después de desplazar una de ellas una distancia t [29].

$$h(t) = (f * g)(t) = \int_{-\infty}^{+\infty} f(\tau) \cdot g(t - \tau) d\tau \quad (3.1)$$

A partir de este concepto, se puede establecer sobre el tratamiento de imágenes una versión discretizada de Ec. 3.1,

$$h(n) = (f * g)(n) = \sum_{i=-\infty}^{+\infty} f(i) \cdot g(n - i).$$

Esta última es ampliable a 2 dimensiones para tratar las funciones matriciales que representan las imágenes. Manteniendo la misma notación se llega a la definición de Ec. 3.2.

Definición. Dada una matriz $F_{m \times n}$ y una matriz $G_{(2N+1) \times (2N+1)}$ con $(2N+1) < m, n$ se define la convolución de las matrices F y G como una nueva matriz $H = F * G$ definida a partir de la expresión

$$h_{ij} = \frac{1}{c} \sum_{r=1}^{(2N+1)} \sum_{s=1}^{(2N+1)} f_{(i-N+r-1), (j-N+s-1)} \cdot g_{r,s}, \quad (3.2)$$

donde $c = \sum_{i,j=1}^{(2N+1)} g_{i,j}$ (si $c = 0$, se toma $c = 1$). Obsérvese que $h_{i,j}$ sólo está definido para $i = (N+1), \dots, (m-N-1)$ y $j = (N+1), \dots, (n-N-1)$ [10, 16, 26]. La fracción se incluye debido a que algunas máscaras pueden saturar la información de cada píxel y no realizar el proceso adecuado.

La matriz G se denomina núcleo o kernel de la convolución y define un filtro particular según los valores que tenga la matriz. En Fig. 3.15 se muestra el proceso de convolución de una matriz dada por un kernel de orden 3x3 y se recoge el resultado correspondiente a la entrada (1, 1) de F . El resultado de este cálculo se obtiene multiplicando los elementos de las matrices uno a uno y sumando los resultados:

$$\begin{aligned} h_{(1,1)} &= \frac{1}{1}(0 \cdot 4) + (0 \cdot 0) + (0 \cdot 0) + (0 \cdot 0) + (1 \cdot 0) + \\ &\quad (1 \cdot 0) + (0 \cdot 0) + (1 \cdot 0) + [2 \cdot (-4)] = -8 \end{aligned}$$

Después de esto, el kernel se desplaza una posición a la derecha sobre la matriz de entrada y se calcula el siguiente píxel. De esta forma, recorriendo todos los elementos de la matriz de entrada, se consigue una matriz de salida en la cual los bordes no pueden calcularse y deben tratarse de forma diferente.

Para el filtrado de imágenes se usa habitualmente matrices kernel de orden 3x3 o 5x5 y a dichas matrices se les denomina también máscaras. Depende de los valores de la máscara que se apliquen la finalidad del filtro es diferente. Existen filtros de detección de bordes (*Sobel*), filtros de enfoque (*Sharpen*), filtros de suavizado (*Gaussiano*), filtros de repujado, etc. [10, 26, 30].

En este trabajo se van a implementar los siguientes kernels:

- 1) Detección de bordes, ‘*Laplaciano*’ con punto aislado.
- 2) Detección de bordes, máscara tipo ‘*Laplaciano*’.

$$G = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} \quad G = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

- 3) Detección de bordes, máscara tipo ‘*Sobel*’ horizontal.

$$G = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

- 4) Detección de bordes, máscara tipo ‘*Sobel*’ vertical.

$$G = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

- 5) Filtro paso alto, máscara tipo ‘*Sharpen*’ o de realzado de bordes (enfoque/nitidez).

$$G = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

- 6) Filtro paso bajo, máscara tipo ‘*Gaussiano*’ o de suavizado.

$$G = \frac{1}{16} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

- 7) Filtro Norte.

$$G = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ -1 & -1 & -1 \end{pmatrix}$$

- 8) Filtro Este.

$$G = \begin{pmatrix} -1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & 1 & 1 \end{pmatrix}$$

3.3.2. Bordes de la imagen

Cuando el kernel está procesando los píxeles en el borde de alrededor de la imagen ocasiona un problema, ya que le faltan píxeles para realizar el cálculo. Por lo tanto, los píxeles de los bordes deben tratarse de forma diferente a los del interior de la imagen [30]. Para ello, existen varias alternativas, algunas de las cuales son:

1) Recortar. Los píxeles del borde se excluyen de la convolución de la imagen, la imagen de salida es más pequeña que la imagen de entrada, Fig. 3.16. La convolución no se realiza en los píxeles del borde porque algunos de los valores no existen y esta solo se realizará cuando el núcleo tenga valores para todos los coeficientes.

2) Frontera Extendida. Los píxeles del borde de la imagen de entrada se extienden para crear una segunda capa de borde. Como se muestra en Fig. 3.17, los píxeles de la esquina de la imagen se copian a los tres píxeles en la capa de borde adicional. Esta solución requiere más lógica para crear la segunda capa de borde.

3) Rellenar con ceros. Versión más sencilla del borde extendido, todos los píxeles fuera de la imagen se tratan como ceros. Fig. 3.18 ilustra cómo funciona esta técnica. Este es el método más sencillo de implementar, si el kernel no puede recuperar un valor de píxel se pondrá a cero.

4) Envolvente. Cuando se procesa un píxel situado en el borde, los valores de píxeles de fuera del borde se toman de la fila siguiente o anterior. En Fig. 3.19 se ilustra un ejemplo sencillo, en este ejemplo se calcula el valor de OP_{10} . Se puede observar que al llegar al final

de la fila se toma la columna de una fila por debajo.

P_1	P_2	P_3	P_4	P_5
P_6	P_7	P_8	P_9	P_{10}
P_{11}	P_{12}	P_{13}	P_{14}	P_{15}
P_{16}	P_{17}	P_{18}	P_{19}	P_{20}
P_{21}	P_{22}	P_{23}	P_{24}	P_{25}

	OP_1	OP_2	OP_3	
	OP_4	OP_5	OP_6	
	OP_7	OP_8	OP_9	

Fig. 3.16: Ejemplo de recorte (*cropping*).

P_1	P_1	P_2	P_3	P_4	P_5	P_5
P_1	P_1	P_2	P_3	P_4	P_5	P_5
P_6	P_6	P_7	P_8	P_9	P_{10}	P_{10}
P_{11}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{15}
P_{16}	P_{16}	P_{17}	P_{18}	P_{19}	P_{20}	P_{20}
P_{21}	P_{21}	P_{22}	P_{23}	P_{24}	P_{25}	P_{25}
P_{21}	P_{21}	P_{22}	P_{23}	P_{24}	P_{25}	P_{25}

OP_1	OP_2	OP_3	OP_4	OP_5
OP_6	OP_7	OP_8	OP_9	OP_{10}
OP_{11}	OP_{12}	OP_{13}	OP_{14}	OP_{15}
OP_{16}	OP_{17}	OP_{18}	OP_{19}	OP_{20}
OP_{21}	OP_{22}	OP_{23}	OP_{24}	OP_{25}

Fig. 3.17: Ejemplo de extensión de frontera (*extended border*).

0	0	0	0	0	0	0
0	P_1	P_2	P_3	P_4	P_5	0
0	P_6	P_7	P_8	P_9	P_{10}	0
0	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	0
0	P_{16}	P_{17}	P_{18}	P_{19}	P_{20}	0
0	P_{21}	P_{22}	P_{23}	P_{24}	P_{25}	0
0	0	0	0	0	0	0

OP_1	OP_2	OP_3	OP_4	OP_5
OP_6	OP_7	OP_8	OP_9	OP_{10}
OP_{11}	OP_{12}	OP_{13}	OP_{14}	OP_{15}
OP_{16}	OP_{17}	OP_{18}	OP_{19}	OP_{20}
OP_{21}	OP_{22}	OP_{23}	OP_{24}	OP_{25}

Fig. 3.18: Ejemplo de relleno de ceros (*zero-padding*).

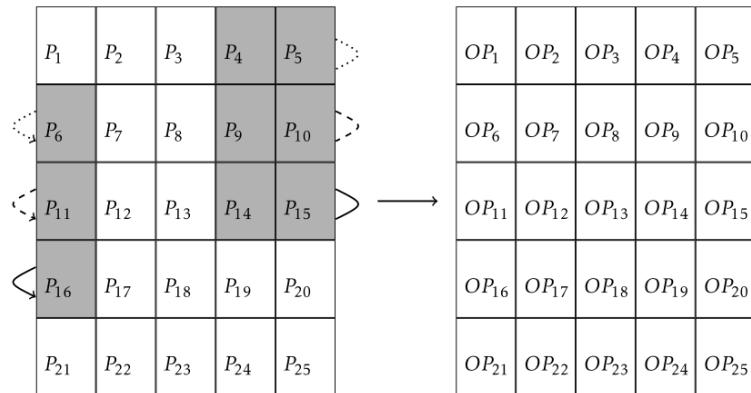


Fig. 3.19: Ejemplo de envolvente (*wrap-around*).

El método *wrap-around* también tiene sus desventajas. Por ejemplo, las filas superior e inferior tienen que usar píxeles del lado opuesto de la imagen, lo cual resulta en un método de direccionamiento más complejo. Una alternativa más fácil es usar relleno cero o extensión de borde en la fila superior e inferior.

Capítulo 4

Arquitectura del sistema

En este capítulo se pretende realizar una descripción del diseño propuesto para abordar este trabajo. Como ya se ha ido comentando en los capítulos anteriores, el sistema consta de tres partes bien diferenciadas. A pesar de tratarse de módulos independientes, el diseño de cada uno de ellos depende también de las especificaciones de los demás, de forma que puedan sincronizarse y ajustarse a los tiempos y estados de cada etapa. Para abordar la explicación completa, se sigue el mismo camino que posee el flujo de datos en el sistema, desde que se captura la imagen con el sensor OV7670, hasta que se muestran por pantalla mediante VGA.

El diagrama de bloques de la arquitectura del sistema completo puede verse en Fig. 4.1. En términos generales, el sistema contiene una cámara de fotos que recibe la información de la configuración interna que debe tener por un puerto serie (protocolo SCCB) desde el controlador. Esta manda los datos de los píxeles de cada fotograma a la FPGA, para que estos se reciban según el protocolo de color *RGB444* y se guarden de forma ordenada en una memoria RAM de doble puerto.

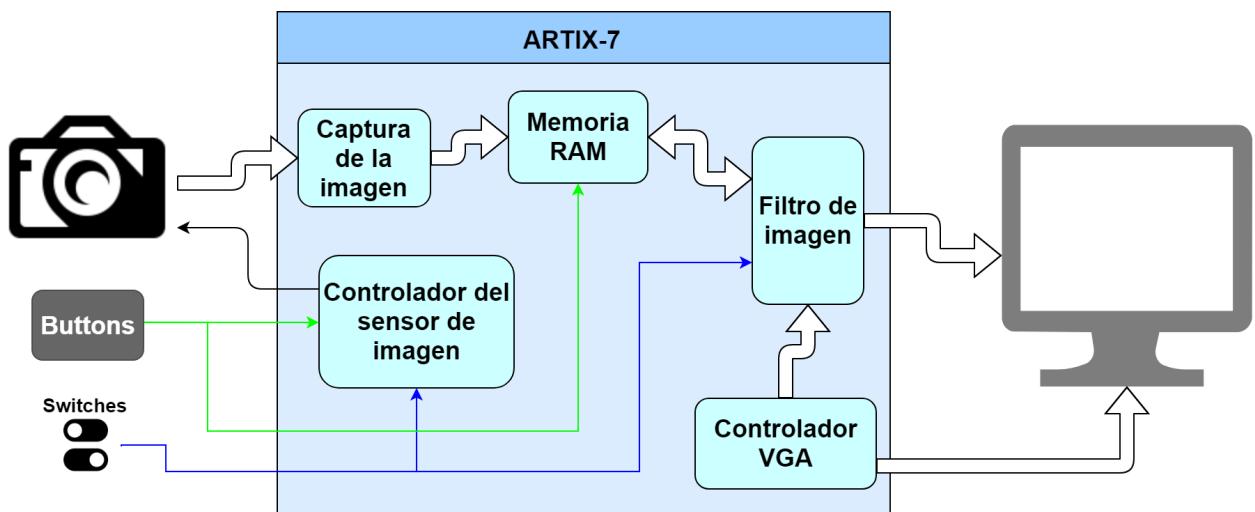


Fig. 4.1: Diagrama de bloques ilustrativo del sistema completo.

Tanto los botones como los interruptores externos a la FPGA se encuentran integrados en la Nexys 4 DDR y pueden configurarse desde Vivado. Se utiliza el botón *btnc*, que va a la memoria, para guardar el fotograma recibido y el botón *btnr* que refresca los parámetros del sensor de imagen (como si fuese un botón para enfocar). El *btnr* no va directamente al controlador, porque antes pasa por un subsistema anti-rebote, pero básicamente tiene la misma función.

También se emplea el interruptor *sw(15)* en el controlador del sensor de imagen para establecer dos estados en la cámara, encendida o suspendida. Este mismo interruptor se usa en el filtro de imagen como variable de control de los tres multiplexores de la salida de datos que van al monitor, para permitir un modo de pantalla apagado. El interruptor *sw(14)* sirve para mostrar en la pantalla la imagen original o la imagen filtrada, y el resto de interruptores sirven para elegir una máscara de filtrado u otra.

La información guardada en la memoria RAM es la que se utiliza para aplicar el filtro de imagen. El sistema de filtrado le manda las posiciones de los píxeles que necesita para obtener la matriz de entrada, F en Ec. 3.2, y poder aplicar el algoritmo. El controlador VGA marca un orden en el que se muestran los datos de cada píxel en la pantalla, en función de a qué píxel estén apuntando sus contadores.

4.1. Sistema de comunicaciones con el OV7670

En el apartado 3.2.1 se realiza una explicación detallada del funcionamiento del sensor que se utiliza en el proyecto y se analizan las partes internas del SoC, así como las señales de comunicación que ofrece la placa adquirida. En esta sección se detalla cómo se ha realizado la implementación del sistema que se comunica desde la FPGA con el sensor de imagen sobre la interfaz Pmod (ver apartado 3.1.2).

Para la creación de este sistema se han realizado una serie de búsquedas en internet y se han utilizado algunos recursos proporcionados por algunos desarrolladores, además de guías de implementación del sensor y otros artículos o trabajos [24, 25, 31, 32, 33, 34].

Al final de la explicación de los distintos módulos se incluye una imagen de la interconexión de todas las partes de este sistema, Fig. 4.8.

4.1.1. Bloque de captura

Para la recepción de los datos procedentes del sensor hay que establecer en primer lugar en qué formato de color se desean recibir los datos. Como las salidas que nos proporciona la Nexys 4 DDR para las señales de color del puerto VGA son de 4 bits, se utiliza el formato RGB444 para utilizar la señal completa que se recibe por la cámara, (Tabla 3.4). Los datos llegan sincronizados con las señales *pclk*, *href* y *vsync*, Fig. 3.13. Dichas señales son las que, junto con *d[7:0]*, forman las entradas de este bloque, Fig. 4.2. La señal *pclk* es la que controla

cuándo se debe leer un dato, mediante los flancos ascendentes; es el reloj del módulo. Además de ello, cuando $vsync = 0$ y $href = 1$, se reciben bytes válidos.

Un aspecto importante a tener en cuenta es que la frecuencia a la que la cámara envía los píxeles es diferente a la que emplea el sistema de filtrado para procesar los datos y a la que emplea el VGA para mostrarlos en pantalla. Por tanto, se debe emplear un espacio de memoria (`capture_memory`) para almacenar los píxeles recibidos y sincronizar el proceso.

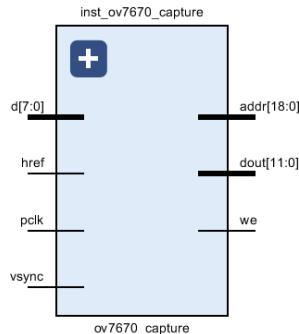


Fig. 4.2: Bloque de captura de imagen.

*Nota: Las entradas se encuentran en el lado izquierdo de los bloques y las salidas se encuentran en el lado derecho (detalle a tener en cuenta para la imagen de este bloque y de los siguientes).

De esta memoria se hablará detalladamente en el siguiente apartado, lo único a tener en cuenta ahora es que cuenta con una entrada que indica la dirección de memoria en la que se escribe el dato y otra que indica cuándo se escribe, por lo que se implementa en este módulo una salida, `addr`, que actualice esa posición de memoria con cada píxel recibido y no haya sobreescritura, y otra, `we` (*write enable*), que se active cuando se haya recibido el píxel e indique que se puede escribir en memoria.

Como cada píxel necesita dos ciclos de `pclk` para recibirse de forma completa, tanto `we` como `addr` deben tener en cuenta ese tiempo e ir escribiendo los datos de forma ordenada (posición a posición) en la memoria posterior. En ella se almacena una imagen completa, por lo que hay que reiniciar la posición de memoria cada vez que se recibe un fotograma completo, es decir, cuando `vsync` cambia a nivel alto.

4.1.2. Bloque de memoria

Este bloque de memoria se implementa a la salida del bloque anterior para almacenar los datos que le llegan. Además de guardar la información, actúa como un interfaz que combina dos sistemas con diferentes periodos de reloj, ya que la velocidad con la que se capturan los píxeles (`pclk`) va cambiando y es necesario sincronizar la lectura de los píxeles para el posterior procesamiento y representación en pantalla.

Se sabe que la resolución con la que trabaja el sensor de captura de imagen es 640x480, de ahí se puede obtener el número total de píxeles que posee una imagen ($640 \cdot 480 = 307.200$),

y sabiendo que cada píxel lleva 12 bits de información, se tiene que el tamaño máximo de memoria que se necesita es

$$\text{Memoria} = \frac{307.200 \text{ px} \cdot 12 \text{ bits/px}}{8 \text{ bits/B} \cdot 1024 \text{ B/kB}} = 450 \text{ kB.} \quad (4.1)$$

Puesto que la FPGA que se utiliza en el proyecto posee 607,5 kB de memoria BRAM no habría problema de espacio de memoria en principio.

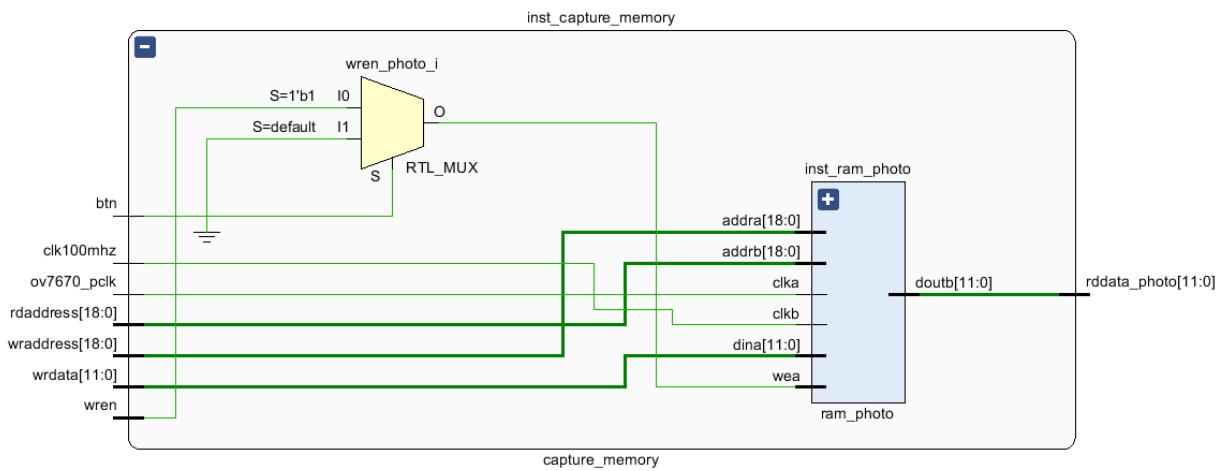


Fig. 4.3: Bloque de memoria.

Para implementar esta memoria se ha utilizado un *Core IP* mediante la herramienta *Block Memory Generator* de Vivado. Como se puede observar en Fig. 4.3, el buffer de memoria se llama “ram_photo”, pero a este se le ha incluido un multiplexor con una entrada de control (*btn*) que maneja la señal *we*, para simular la captura de imágenes como si de una cámara de fotos se tratase. Con los valores de la resolución (profundidad de la memoria) y el número de bits de cada píxel (ancho de palabra) se define la RAM de doble puerto y se configuran los puertos como “*always enabled*”. Este botón se conecta al *btnc*.

El puerto A es el puerto de escritura y las señales que están relacionadas con este son las que proceden del bloque de captura. El reloj de este puerto (*clka*) también se conecta al reloj generado por el sensor de imagen. Esto se realiza así porque, como se ha mencionado, el bloque de captura del apartado anterior es tan solo un adaptador de la señal del sensor de imagen, es decir, se trata de un receptor que ordena los datos de cada píxel en direcciones de memoria independientes y los envía a “ram_photo”, por lo que ambos módulos tienen que estar sincronizados con el mismo reloj.

El puerto B es el puerto que se utiliza de lectura y que tiene otro reloj para separar ambos procesos (lectura y escritura). Este puerto está controlado por el sistema de filtrado que es el que utiliza los resultados de la captura de imagen para el posterior procesado. Para la configuración de este puerto se indica la opción de “*write first*” para que no haya problemas con la pérdida de información por parte de la señal de entrada en el puerto A. De todos

modos, eso no sería un problema en este sistema porque cuando se captura una imagen, esta no continúa cambiando hasta que se vuelva a capturar otra.

4.1.3. Bloque de control

Una vez que ya se tienen definidos los métodos de almacenamiento de imagen y los espacios para ello, se configura el sensor mediante el protocolo de comunicación que soporta, el SCCB⁷, que como ya se ha mencionado en el apartado 3.2.1 es muy similar al protocolo I^2C .

En consecuencia, se debe diseñar un sistema de comunicación que actúe de maestro, para configurar la placa, la cual será el esclavo. Existen dos señales para crear el protocolo de comunicación que son *sioc* y *siiod*, la señal de reloj y de datos respectivamente. Leyendo la hoja de especificaciones del protocolo SCCB [35], se recomienda que para la señal de *sioc* la frecuencia sea como máximo 100 kHz. Esto implica que hay que realizar también un divisor de frecuencia.

Además de esta comunicación, el módulo de control maneja también el resto de señales que el sensor de imagen necesita recibir para funcionar correctamente, que son:

→ ***xclk***: se configura a 25 MHz, debe mantenerse entre 10 y 48 MHz [24].

→ ***pwdn***: se conecta a un interruptor de la Nexys 4 DDR, en concreto al *sw(15)*, que será el mismo interruptor que se utilizará para poner la pantalla en negro y establecer un modo de suspensión en ambos dispositivos.

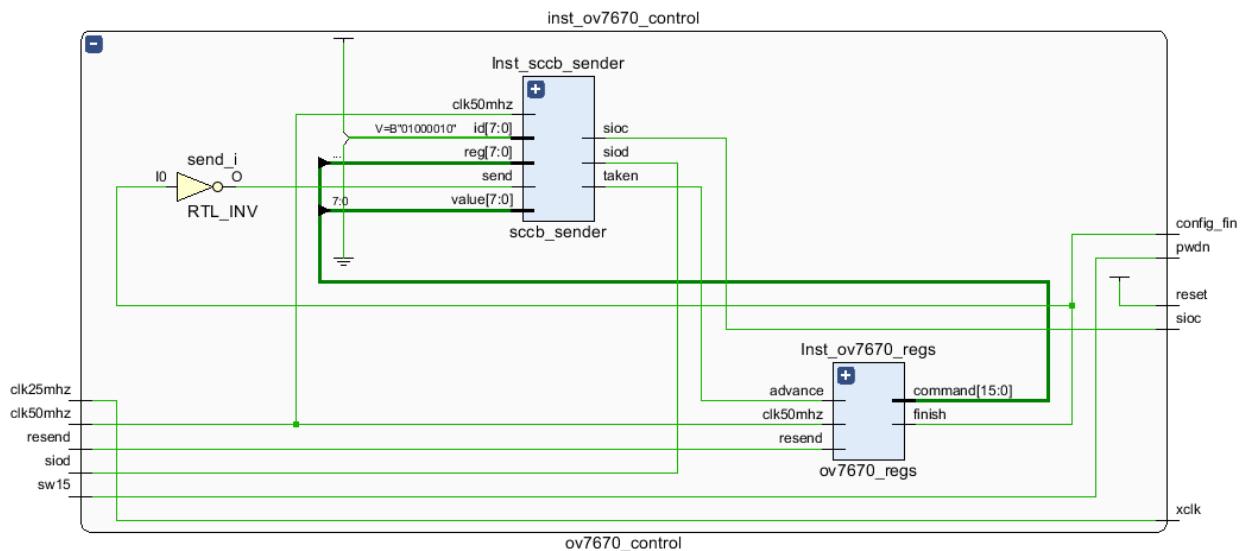


Fig. 4.4: Bloque de control del OV7670.

⁷Es un protocolo de comunicación que trabaja sobre un bus serie de 3 hilos y está diseñado por OmniVision para el control de la mayoría de funciones de su gama de sensores de imagen. En el caso de la placa de la que se dispone, la versión es de 2 hilos.

→ **reset**: se configura a nivel alto, para no limpiar los registros. Sobre estos se hace un reset software utilizando un mensaje en la serie de datos enviados a través de *siod*: en la dirección 0x12 se escribe el valor 0x80.

En Fig. 4.4 se muestra la configuración interna de este módulo. Como se observa, se diseñan dos subsistemas que facilitan la creación del control completo. También se añade una señal de salida extra, *config_fin*, que se conecta a un LED para indicar que la configuración del sensor ya se ha realizado.

4.1.3.1. Bloque emisor SCCB

Este subsistema es el encargado de implementar el protocolo de comunicación. Para su implementación es necesario entender con detalle cómo hay que configurar cada estado del protocolo y construir los paquetes de mensajes de forma adecuada [35].

Para comenzar, se parte de que la señal *sioc* se mantiene en estado alto para mantener un estado de reposo en la comunicación y que en los flancos de bajada se realizan las transmisiones de los datos. Durante el tiempo de reposo, *siod* se mantiene en alta impedancia, ya que al tratarse de una señal bi-direccional puede ser escrita por ambos dispositivos. Una vez dicho esto, para la transmisión de los mensajes se pueden distinguir tres etapas: inicio de la transmisión, cuerpo de la transmisión y fin de la transmisión.

- **Inicio de la transmisión.**

Como Fig. 4.5 muestra, cuando *siod* se cambia de alta impedancia a estado alto y seguidamente pasa a estado bajo, comienza la transmisión de un mensaje. Mientras esto ocurre *sioc* se debe mantener en estado alto. Es el máster el único que puede iniciar la comunicación mediante este procedimiento y podrá leer o escribir hasta que se finalice la transmisión o se inicie otra.

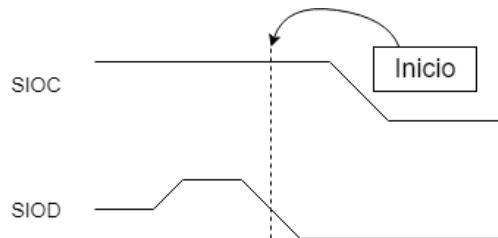


Fig. 4.5: Inicio de la transmisión en SCCB de 2 cables.

- **Cuerpo de la transmisión.**

En la transmisión de los mensajes, hay un elemento básico que se denomina fase, con el que se juega para crear tres tipos de transmisiones de lectura y escritura, pero esta explicación

solo se va a centrar en la transmisión de mensajes de escritura, que es el tipo de mensajes que se utiliza. En Fig. 4.6 se muestra la estructura de un mensaje, el cual tiene 3 fases de 9 bits cada una y en el que se manda el MSB primero. En las tres fases, el noveno bit no afecta, es decir, es un bit que solo sirve para separar las fases y su valor no se tiene en cuenta.

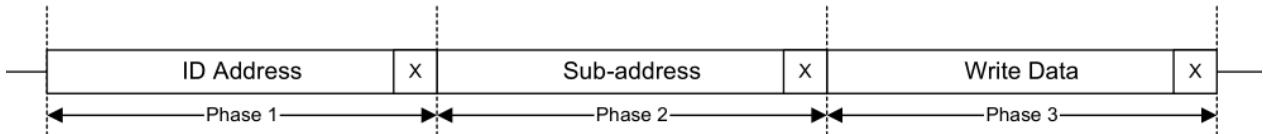


Fig. 4.6: Estructura de un mensaje en SCCB de 2 cables.

La fase 1 se utiliza para identificar al esclavo que recibe el mensaje. En los 8 bits útiles del mensaje de esta fase, 7 bits son de datos: para identificarlo; y 1 bit (el octavo): para indicar el sentido de la transmisión entre el máster y el esclavo, si el mensaje es de escritura se pone a ‘0’ y si es de lectura se pone a ‘1’. En particular para el OV7670, se mantiene constante al valor hexadecimal 0x42, que se corresponde con el identificador de escritura sobre este esclavo (ver Fig. 4.4).

La fase 2 indica la dirección del esclavo en la que se pretender escribir, es decir, la dirección del registro interno del sensor de imagen. Para este sensor van desde 0x00 a 0xC9.

La fase 3 contiene el valor que se desea almacenar en el registro, indicado en la dirección de la fase 2 y en el esclavo identificado por la fase 1.

- **Fin de la transmisión.**

Tiene exactamente la estructura opuesta a la del inicio de transmisión, respetando que *sioc* debe estar en estado alto, basta con que *siod* marque una subida a estado alto y ya mantenga el estado de alta impedancia, para que se marque la finalización de la transmisión.

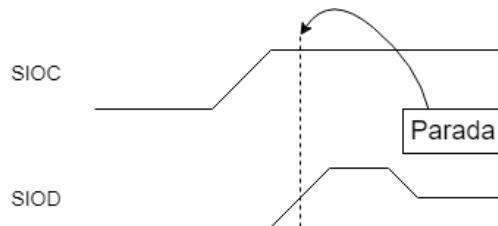


Fig. 4.7: Fin de la transmisión en SCCB de 2 cables.

Teniendo en cuenta para la implementación todo esto que se ha explicado, en Fig. 4.4 se puede ver el resultado del bloque “*sccb_sender*” que se ha diseñado. Este módulo cuenta con una señal de entrada, *send*, (esta señal es la negada de la salida que va al pin del LED, *config_fin*) para comenzar con la transmisión de los valores de los registros que se desean mandar, cuando el LED se activa significa que se han enviado los datos de los registros y que *send* se pone en estado bajo bloqueando la transmisión de más mensajes.

Ya se ha comentado que *id* se mantiene en un valor constante para comunicarse con el esclavo en modo escritura (0x42). También se ha creado una señal de comunicación con el bloque de registros, *taken*, que se pone en estado alto para avisar de que ya ha finalizado un envío y que se actualice al siguiente mensaje a enviar.

Se utiliza una señal interna, *divider*, para crear el contador que divide la frecuencia de la señal de reloj de 50 MHz y se ajusta a la que debe tener *sio*. En la imagen del bloque se puede apreciar que *siod* es una salida y se conecta al bloque de control externo como entrada, esto se debe a que se ha configurado en la implementación como una señal “*inout*”.

4.1.3.2. Bloque de registros

La implementación de este bloque es muy sencilla ya que se limita a guardar los valores que se desean para cada registro que se puede escribir (ver Tabla 8-2 en [25]).

Se trata de la arquitectura de una memoria ROM, en la que se tiene una señal de entrada *advance*, que permite incrementar la señal interna *address*, que se comporta como un contador de programa, para mandar el siguiente valor con su dirección de registro (*command*). Esta señal *advance* es la que viene conectada del emisor, *taken*, que ya se ha comentado en el apartado anterior, Fig. 4.4.

La otra señal de entrada que no es el reloj del bloque, es *resend*, que viene conectada desde el exterior y sirve para reiniciar el “contador de programa” y el envío, es decir para volver a mandar la configuración de los registros. Para esta señal de reinicio se utiliza el botón externo *btnr* al que, como ya se ha comentado, se le realiza una implementación de un sistema anti-rebote.

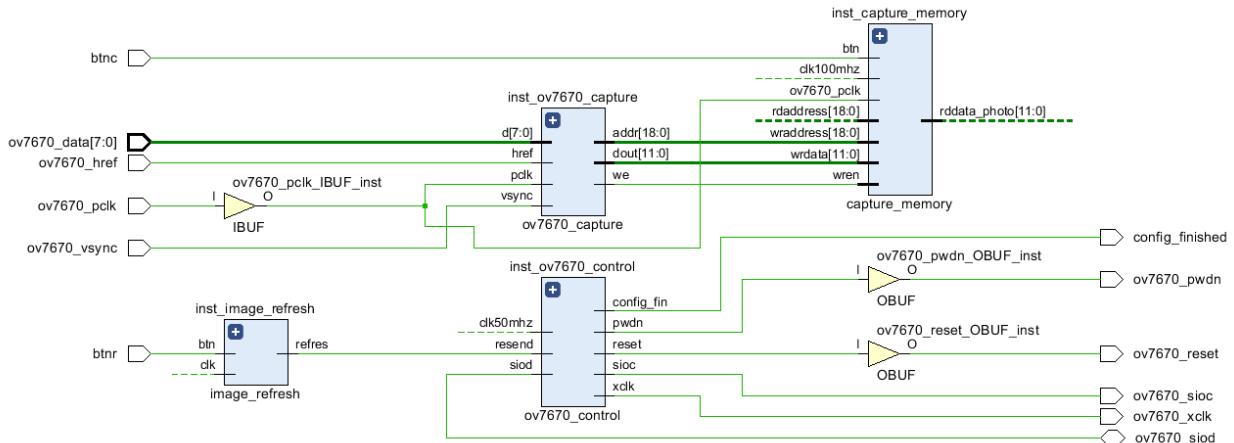


Fig. 4.8: Esquema de interconexión de cada uno de los bloques del sistema de captura de imagen y configuración del sensor OV7670.

Finalmente, la interconexión de los bloques que se han descrito en esta sección se puede observar en Fig. 4.8. Se puede apreciar que este sistema de captura de imagen es independiente

de los posteriores y, como ya se ha mencionado, se encarga del control del sensor de imagen y de sacar por *rddata_photo* los datos de cada píxel de la imagen de forma ordenada.

El bloque que no se ha explicado, “image_refresh”, tiene la función de actualizar los valores de los registros de la cámara, es decir, volver a realizar el proceso de envío de los valores de los registros que se han configurado, con el pertinente reset software (envío del valor 0x80 al registro de la dirección 0x12), para reajustar automáticamente condiciones de luminosidad y brillo en el sensor de imagen. Por ejemplo, para adaptarse a un cambio de incidencia de la luz en el lugar donde enfoca la cámara o situaciones similares, es muy útil para reajustar la intensidad de color y la calidad de la imagen previamente a ser capturada. El botón que realiza esta función es el botón *btnr*.

4.2. Sistema de filtrado

Tal y como se ha explicado en el Capítulo 3, en este sistema de filtrado se realiza un proceso de convolución de matrices. El bloque resultante de la implementación se muestra en Fig. 4.9. Antes de explicar la implementación, se detallan los funcionamientos de las entradas y las salidas.

Por un lado, a parte de las señales de los relojes y el reset, las entradas al sistema son:

→ ***rddata_photo***: esta señal se encarga de recibir los datos de cada píxel que se han guardado en el interior de la memoria y que se van solicitando con *rdaddress*. Hay que tener en cuenta para el diseño que la latencia de la memoria RAM es de 2 ciclos de reloj y que los píxeles se guardan en ella ordenados por columnas y filas, en este orden: (0,0) ... (639,0) (0,479) ... (639,479) (utilizado una ilustración como Fig. 4.10 se entiende de forma más clara).



Fig. 4.9: Bloque de filtrado de imagen.

Por ello, las direcciones de los valores de cada píxel que se tengan que traer de memoria

en cada instante, se calculan como una combinación de las coordenadas en ese instante: $rdaddress = pixel_y \cdot 640 + pixel_x$. Lo que hay que tener presente es que las señales de las coordenadas son los punteros de cada eje sobre la pantalla.

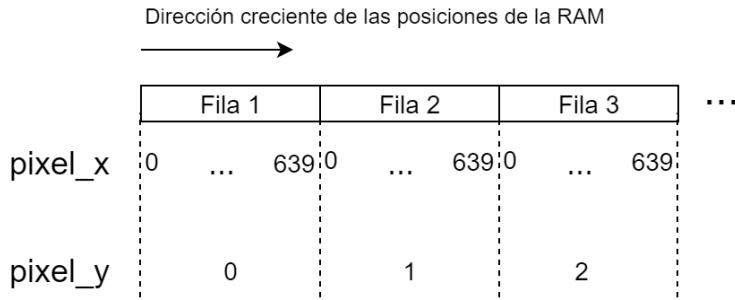


Fig. 4.10: Ilustración que ejemplifica los valores de las direcciones en RAM de cada píxel.

→ ***sw*:** es una señal procedente de los interruptores habilitados en la Nexys 4 DDR y permiten al usuario interactuar con el sistema, en este caso, para seleccionar las distintas máscaras de filtrado que se configuran en el interior. El interruptor *sw*(14) se utiliza para mostrar la imagen original o la imagen filtrada. Los interruptores del 0 al 7 eligen entre las diferentes máscaras que se implementan, en el apartado 3.3.1 se detalla cada una de las máscaras que se pueden seleccionar.

→ ***pixel_x*:** es la señal que lleva la cuenta de los píxeles de cada fila, simulando la coordenada en el eje X de la imagen.

→ ***pixel_y*:** es la señal que lleva la cuenta de las filas de la imagen, para simular la coordenada en el eje Y de la imagen.

→ ***area_activa*:** la finalidad de esta señal es apoyar al módulo de filtrado para que sepa en qué momento se tiene disponible la pantalla para representar los píxeles.

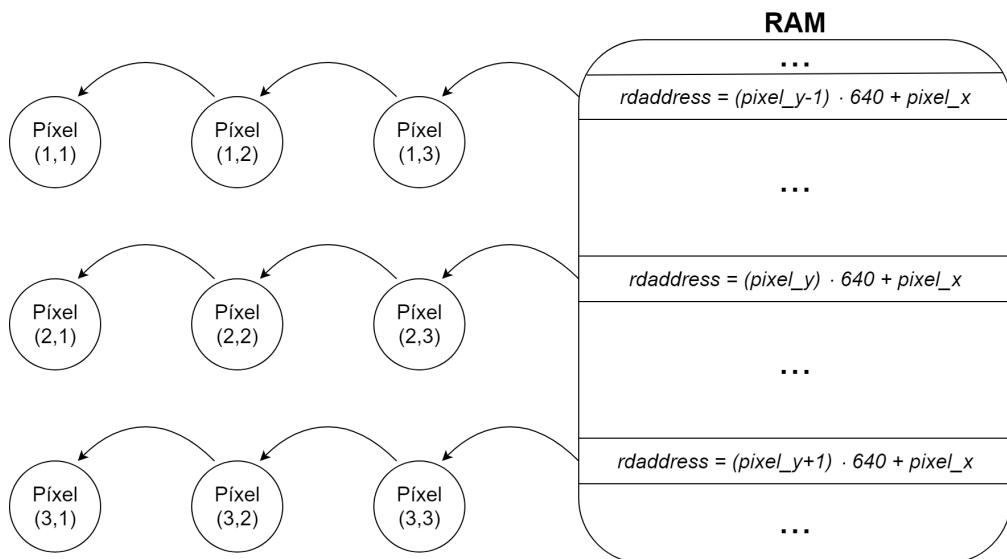


Fig. 4.11: Método de carga de los datos desde la memoria RAM.

Por otro lado, entre las salidas se encuentran las tres señales de color (*data_r*, *data_g*, *data_b*) que van conectadas a la pantalla a través del conector VGA. Para que se mande el valor adecuado que debe tener cada píxel en la pantalla, es importante que los valores de estas tres señales estén coordinados con los valores que tengan las señales *pixel_x* y *pixel_y*. Más adelante se hablará sobre estas señales, ya que son salidas del bloque VGA, simplemente mencionar ahora que son las señales de los contadores y representan las coordenadas de cada píxel sobre la pantalla.

La otra salida que tiene el bloque es la señal de la dirección de los datos (*rdaddress*) que se le piden a la memoria RAM que guarda la foto (“capture_memory”), para ir recorriendo todas las posiciones e ir cogiendo cada pixel de la foto que esta tiene almacenado. Fig. 4.11 es una representación que ilustra el método en el que se ha implementado la carga de los datos desde la memoria. Este método simula la tarea que tiene el kernel de recorrer la imagen; en este caso se realiza al revés, es decir, el kernel se mantiene fijo y se van pasando los datos en el sentido inverso sobre el kernel, de este modo la operación de convolución no se altera. Además de ello, se aprovecha que se recorren los píxeles uno a uno cada ciclo de 25 MHz (debido al controlador VGA) y que se puede utilizar el reloj de 100 MHz para leer datos de la RAM. Como solo hay un bus de lectura para la memoria los datos se tienen que leer de uno en uno y a 100 MHz se pueden leer hasta 4 datos en un ciclo de 25 MHz.

Para la implementación de los cálculos de filtrado, lo primero pasa por diferenciar aquellos píxeles que pertenecen a los bordes de aquellos otros que se ubican en la parte central de la imagen. Esta distinción se ha realizado mediante una máquina de estados, representada en Fig. 4.12. Posteriormente se aplica el algoritmo de convolución, Ec. 3.2, en el estado en el que se leen píxeles internos ('middle_im').

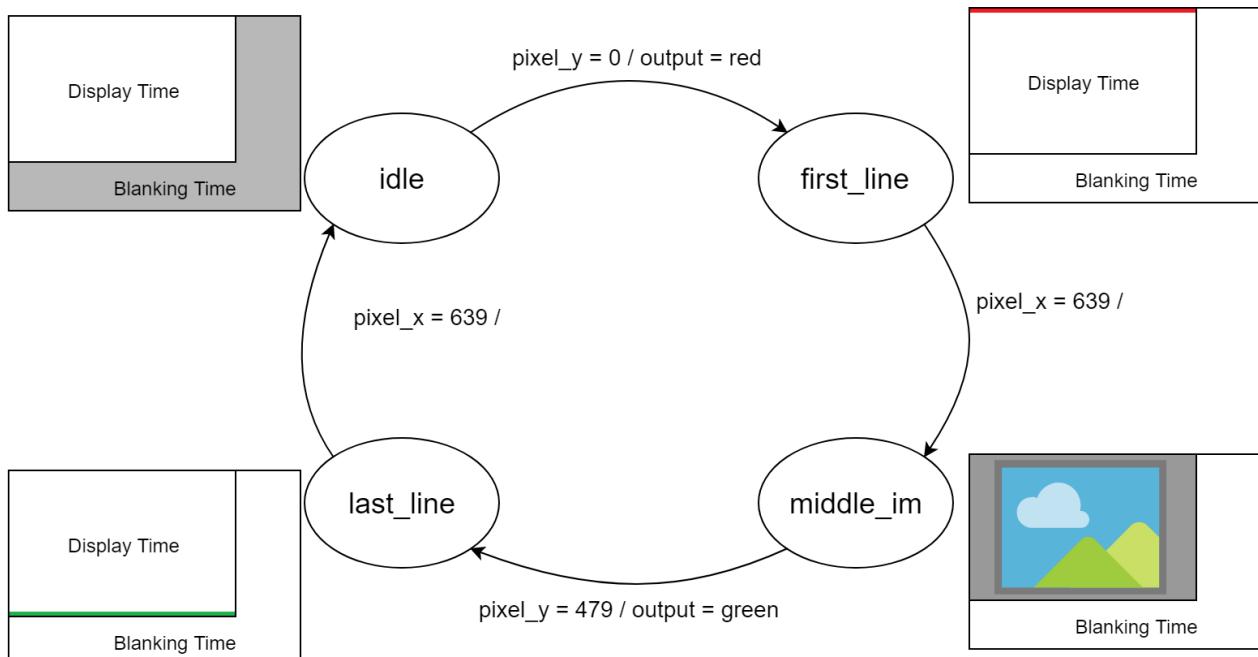


Fig. 4.12: Máquina de estados de la salida de color sobre la pantalla.

La máquina de estados que controla los momentos en los que se tiene que realizar el filtrado de píxeles visualizables, se ilustra en Fig. 4.12 y tiene los siguientes cuatro estados:

- **idle**: este estado sirve como estado de reposo, para no enviar datos que se desean pintar sobre la pantalla en los momentos en los que los haces del monitor no apuntan a la zona visible.

- **first_line**: en este estado se pinta un color constante sobre la pantalla para visualizar la parte alta del marco mientras se observan los resultados del filtrado. Hay diferentes formas de tratar los bordes, ya que sobre estos no se puede aplicar la convolución (apartado 3.3.2). La salida para este estado puede elegirse en función de la aplicación del sistema, para adecuarse a la situación. Se elige un color para depurar.

- **middle_im**: en este estado se procesa la imagen cuando la señal de la coordenada sobre el eje X apunta al interior de la pantalla visible (*area_activa* = 1). Mientras el valor de *pixel_x* se encuentre entre 0 y 639, la salida está conectada a la implementación de Ec. 3.2.

- **last_line**: en este estado ocurre lo mismo que en *first_line*, simplemente se elige otro color para comprobar el correcto funcionamiento de la implementación de la FSM.

Empleando esta FSM se implementa el DFG (Data Flow Graph) de Fig. 4.13 para crear el pipeline de cálculos de datos de salida.

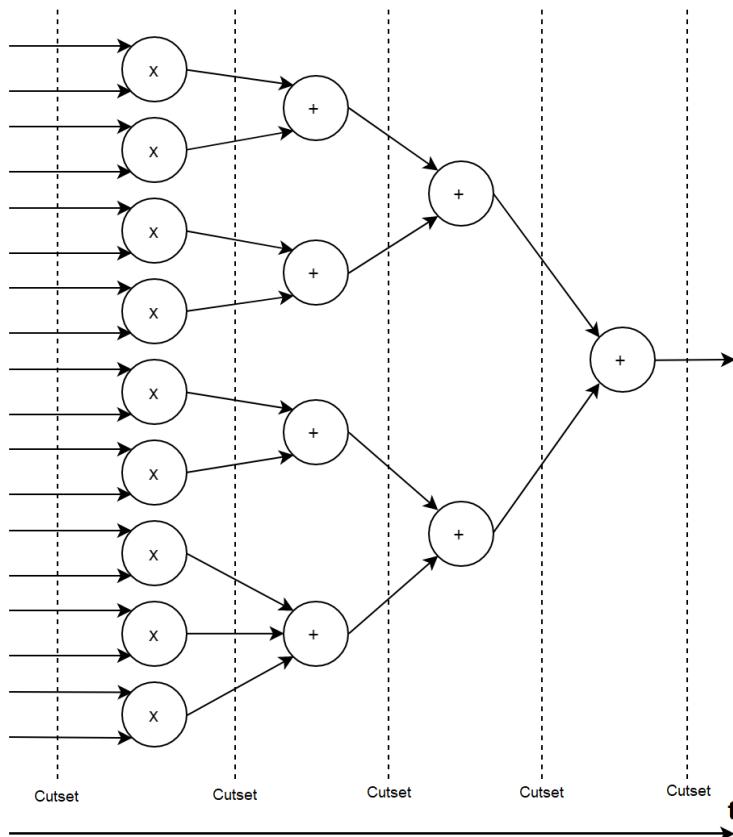


Fig. 4.13: Gráfico del flujo de datos para la implementación de Ec. 3.2.

Este DFG se trata de un pipeline de 4 etapas. De estas, la primera es una etapa consiste en una serie de 9 multiplicadores en paralelo que multiplican la matriz de entrada por el kernel de filtrado. Las tres etapas siguientes componen un árbol de sumas (*addition tree*) paralelizado en espacio y tiempo. Este se encarga de sumar los resultados de los multiplicadores.

Teniendo en cuenta las etapas de conversión de los datos y las de carga de la memoria, se tiene que en total el sistema tiene una latencia de 9 ciclos de reloj.

4.3. Sistema controlador de VGA

Para el diseño del sincronizador VGA, no solo hay que tener en cuenta las explicaciones que se dan en capítulos anteriores, sino también hay que saber cuáles son los tiempos de sincronización, Tabla 4.1, para los contadores que controlan las señales que se le envían al control de desviación del haz y así recorrer toda la pantalla. Hoy en día las pantallas soportan varias resoluciones diferentes, en función de los tiempos de las señales que se le mandan se consiguen unas u otras resoluciones.

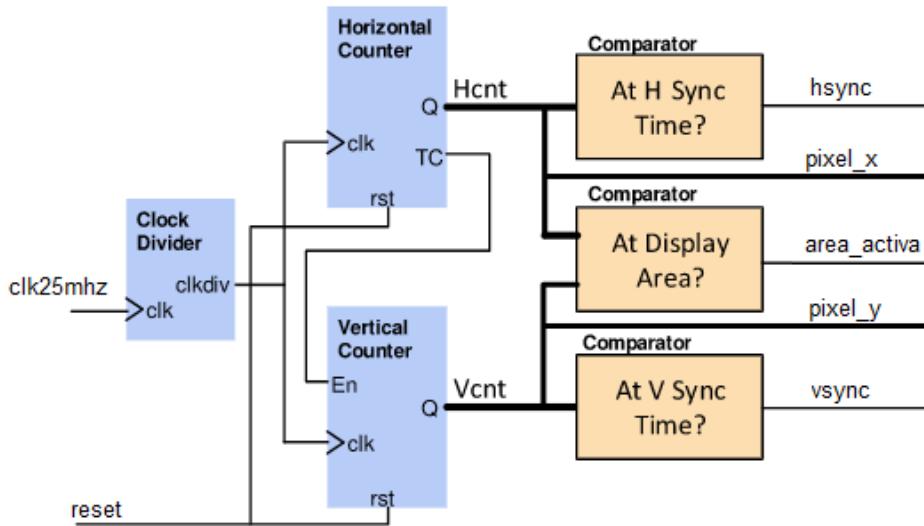


Fig. 4.14: Diagrama de bloques internos del funcionamiento del controlador VGA.

Un circuito controlador de VGA debe generar las señales de sincronización *hsync* y *vsync* en coordinación con la entrega de datos, todo ello basándose en la velocidad de los píxeles. Esta define el tiempo disponible para mostrar un píxel de información y en el caso de la resolución VGA es de unos 25 MHz. Dicho valor se obtiene porque la pantalla tiene 800 píxeles por cada línea y en total hay 525 líneas, por lo que si se quiere refrescar la pantalla a 60 Hz para que el ojo humano no sea capaz de distinguir este refresco, se tiene que la frecuencia por píxel es [36]

$$\text{Pixel rate} = p \cdot l \cdot s = 800 \text{ px/line} \cdot 525 \text{ lines/screen} \cdot 60 \text{ screens/seg} \approx 25 \text{ MHz.}$$

Para implementar este bloque (Fig. 4.14) se necesita una entrada de reloj de 25 MHz, ya que no es necesario ajustarse exactamente al resultado que se obtiene del cálculo anterior. También es necesario realizar dos contadores, uno que lleve la cuenta de los píxeles en una fila y otro que lleve la cuenta de las filas de la pantalla. Con ayuda de los valores de Tabla 4.1, los de la acumulación, se puede saber en qué partes hay que poner en estado alto y bajo las señales de sincronización, *hsync* y *vsync*. En Fig. 4.15 se han representado esos valores, que al final no es más que la suma de los valores de la tabla. De esta forma para implementar *hsync* y *vsync* solo hay que realizar la comparación de esos valores.

Tabla 4.1: Tabla de tiempos y equivalencias en los contadores para VGA.

Parameter	Vertical Sync			Horizontal Sync		
	Time [ms]	Count	Acc	Time [μ s]	Count	Acc
Display (active)	15.36	480	480	25.6	640	640
Front Porch	0.32	10	490	0.64	16	656
Retrace (sync pulse)	0.064	2	492	3.84	96	752
Back Porch	1.056	33	525	1.92	48	800

Hay que tener en cuenta que los contadores son de módulo 525, es decir, van de 0 a 524 y de módulo 800, de igual forma, cuentan de 0 a 799. También son importantes las salidas de los contadores, *pixel_x* y *pixel_y*, porque identifican de forma única sobre qué píxel de la pantalla se encuentra el haz.

De esta forma, y con la ayuda de otra señal que indique cuándo se encuentran los contadores en la parte activa de la pantalla, *area_activa*, se puede indicar el color que debe tener ese píxel, y así poder mostar cualquier imagen sobre la pantalla. Por ello, estas tres últimas señales se le deben enviar al módulo que manda los datos de color a la pantalla para que estén sincronizados.

Con Fig. 4.15 se pretende ilustrar el significado o interpretación que tienen las señales de sincronización sobre la pantalla. Analizando la imagen, se entienden cuáles son los momentos en los que se envía información (parte visible o activa) y los momentos en los que no.

En la ilustración se observa que los contadores comienzan cuando la pantalla se encuentra activa, es decir, empiezan a contar con la pantalla en el pixel (0,0), de modo que se facilita el entendimiento del proceso de implementación. Sin embargo, gracias a las señales de sincronización, la pantalla lo entiende de la misma forma aunque se desplacen los pulsos de sincronización y se imagine que la parte de “Display Time” de Fig. 4.15 no se encuentre en la esquina superior izquierda de la pantalla completa. En el fondo, el refresco de pantalla se trata de un proceso repetitivo y circular, luego no importa desde qué punto de vista se imagine.

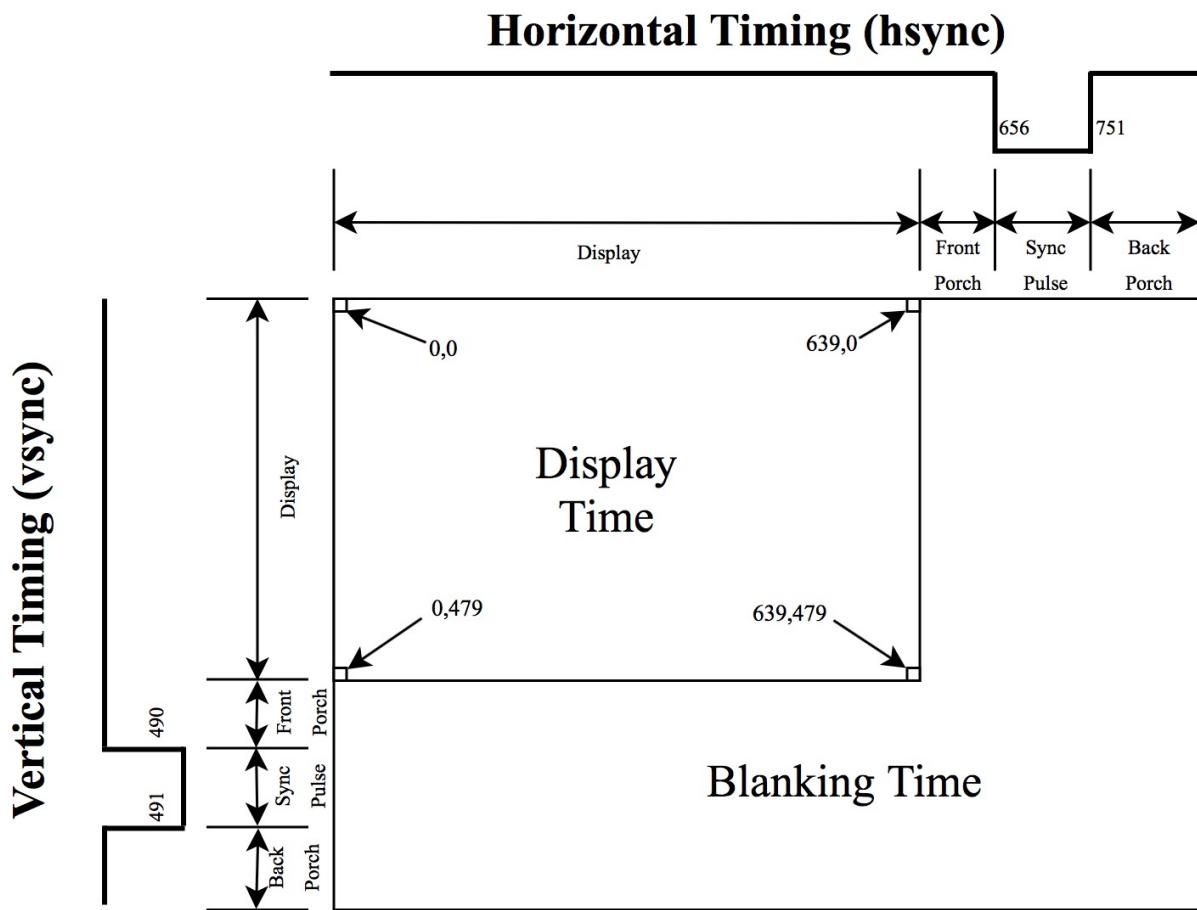


Fig. 4.15: Ilustración de las señales de sincronización VGA sobre una pantalla.

Capítulo 5

Resultados y Discusión

En este Capítulo se pretende realizar una exposición de los resultados obtenidos ante un proceso de filtrado convolucional sobre una imagen pura precargada en la memoria RAM. El objetivo consiste en utilizar un modelo software para verificar los resultados de la aplicación de las máscaras implementadas en hardware. Posteriormente, se pretende comprobar su correcto funcionamiento sobre imágenes a color que se capturen con el sensor de imagen OV7670 o se precargen en memoria y analizar los resultados.

Además, se incluyen explicaciones importantes sobre aspectos de diseño que se han tenido en cuenta para poder corregir los errores en la implementación del sistema. Para ello, se utilizarán los resultados de los *testbench* realizados, incluyendo los cronogramas de tiempo obtenidos de las simulaciones para apoyar dichas explicaciones.

Finalmente, se realiza una discusión de los recursos y el consumo de potencia que esta implementación requiere en una FPGA Artix-7 de Xilinx. Dichos datos son proporcionados por las herramientas que Vivado ejecuta sobre el diseño implementado.

5.1. Resultados de los kernels

En esta sección se tratan los resultados sobre las máscaras ensayadas en el sistema (apartado 3.3.1), haciendo una comparación de estos mismos resultados y los que produce un software de licencia gratuita temporal que permite introducir la máscara que el usuario desee para comprobar su resultado, este es el Corel® PaintShop Pro 2019.

En primer lugar se construye una imagen pura (Fig. 5.1) mediante dicho software para comprobar que las diferentes máscaras realizan el mismo procesamiento en la FPGA y en el programa. Una vez que se tiene la imagen, se guarda en formato ‘.bmp’ y utilizando el software MATLAB® se convierte esta desde el formato original (RGB888) a una imagen en formato hexadecimal (RGB444). Esta se guarda en un archivo ‘.coe’ para cargarla en la memoria mediante Vivado, para ello se le añade una cabecera que le indica el sistema de numeración y el vector de datos a guardar.

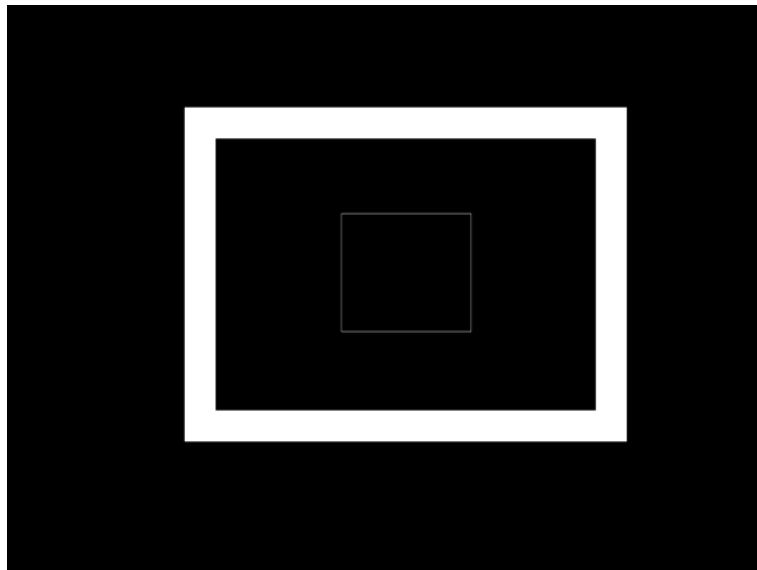
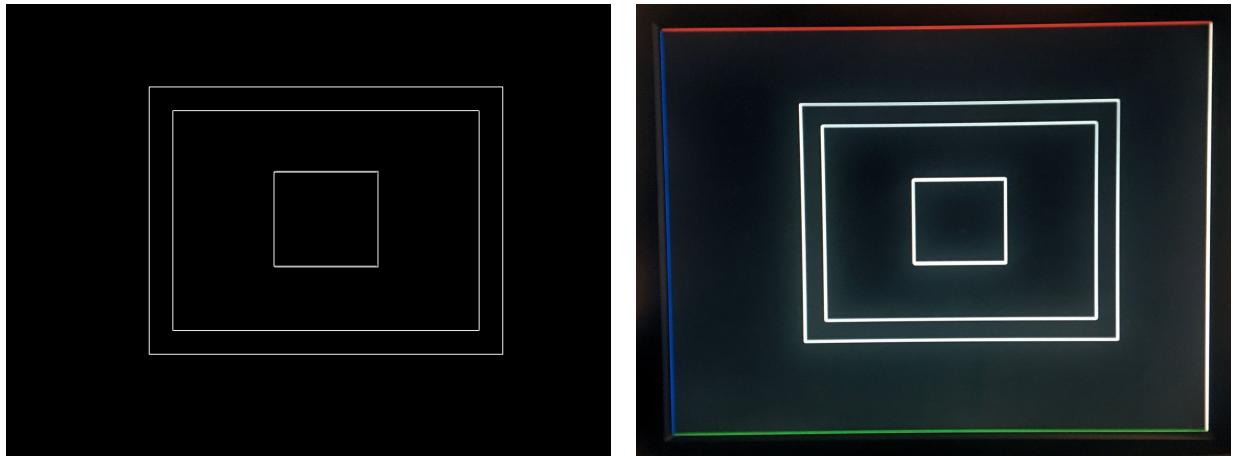


Fig. 5.1: Imagen pura de resolución 640x480.

Esta imagen es la que se ha utilizado para validar y verificar el comportamiento del filtro en la FPGA. También se utilizó MATLAB para calcular el algoritmo de convolución según los valores hexadecimales de los píxeles e ir comparando píxel a píxel con los cronogramas en Vivado la entrada de datos desde la memoria RAM y si el flujo de estos era correcto a cada paso de la señal de reloj de 25 MHz a través del pipeline.

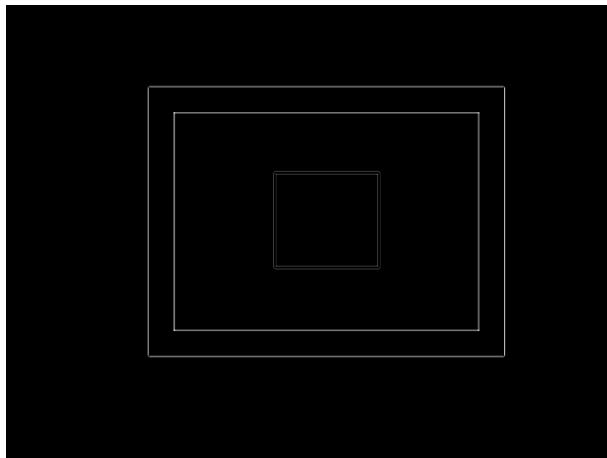
Los resultados se muestran comparados con los del software de edición de imágenes a continuación. Se ha decidido incluir el Apéndice C con los resultados sobre imágenes a color, las cuales muestran de forma más clara la funcionalidad de cada kernel que se aplica. Numéricamente los resultados son iguales en ambas imágenes, pero las imágenes procesadas por la FPGA han sido fotografiadas del monitor CRT.



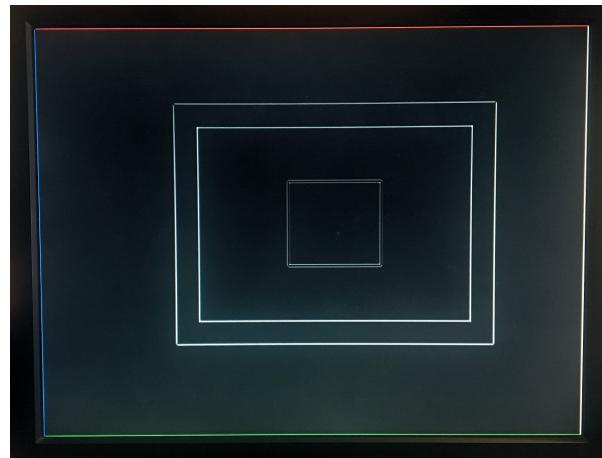
(a) Utilizando PaintShop.

(b) Utilizando la FPGA.

Fig. 5.2: Resultado del filtro con el kernel 1 (detección de bordes 1) sobre Fig. 5.1.

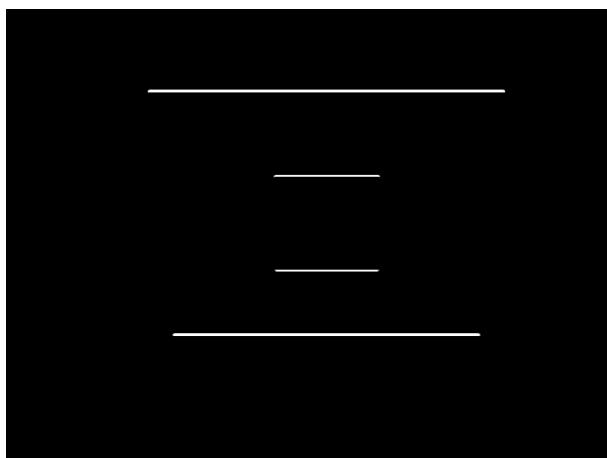


(a) Utilizando PaintShop.

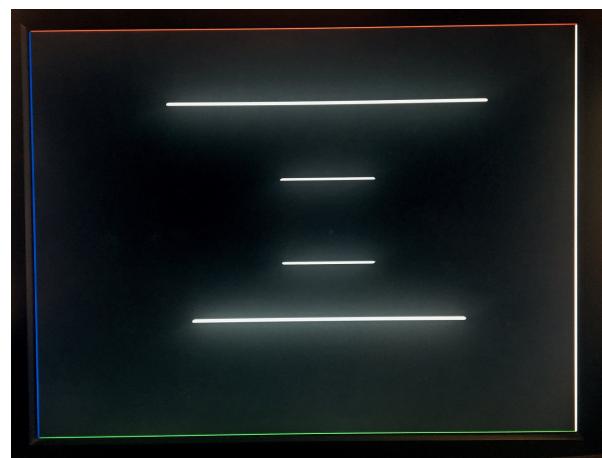


(b) Utilizando la FPGA.

Fig. 5.3: Resultado del filtro con el kernel 2 (detección de bordes 2) sobre Fig. 5.1.

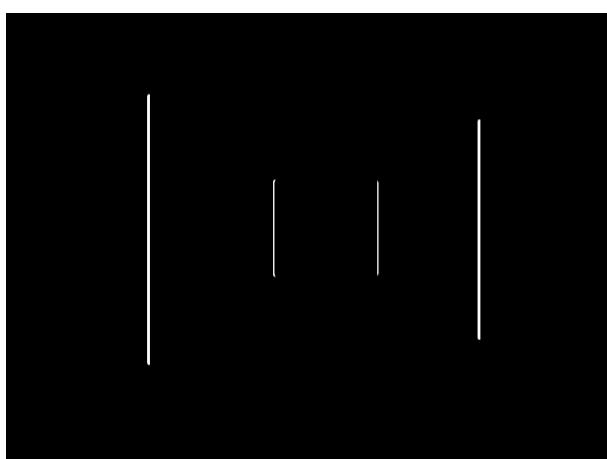


(a) Utilizando PaintShop.

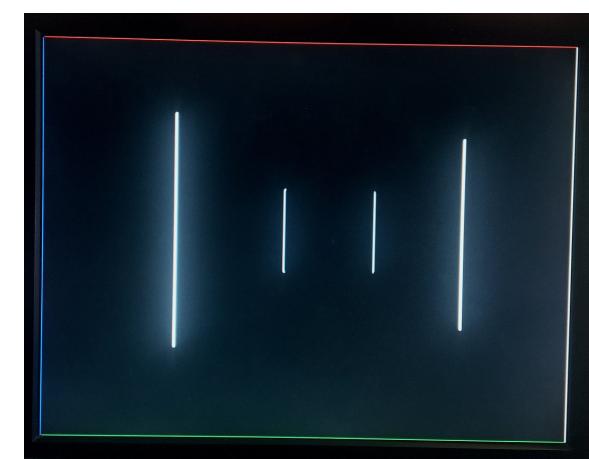


(b) Utilizando la FPGA.

Fig. 5.4: Resultado del filtro con el kernel 3 ('Sobel' horizontal) sobre Fig. 5.1.

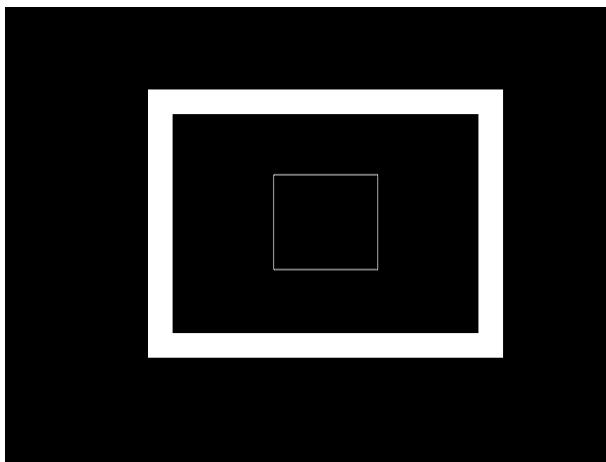


(a) Utilizando PaintShop.

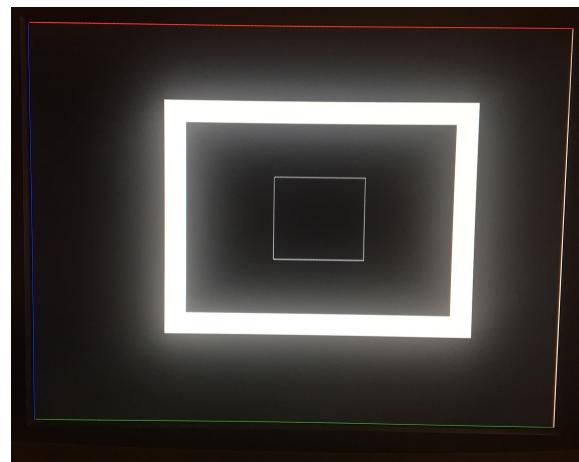


(b) Utilizando la FPGA.

Fig. 5.5: Resultado del filtro con el kernel 4 ('Sobel' vertical) sobre Fig. 5.1.

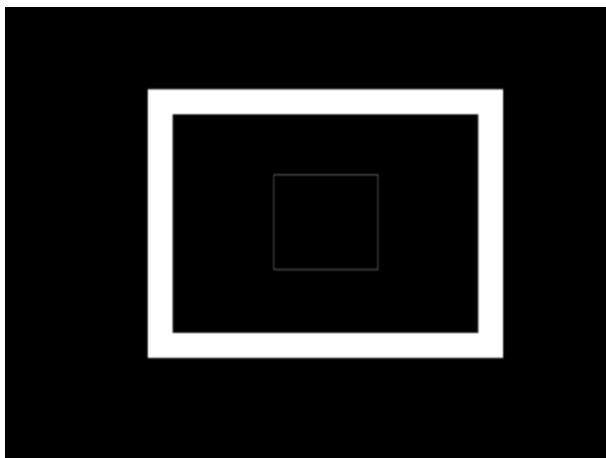


(a) Utilizando PaintShop.

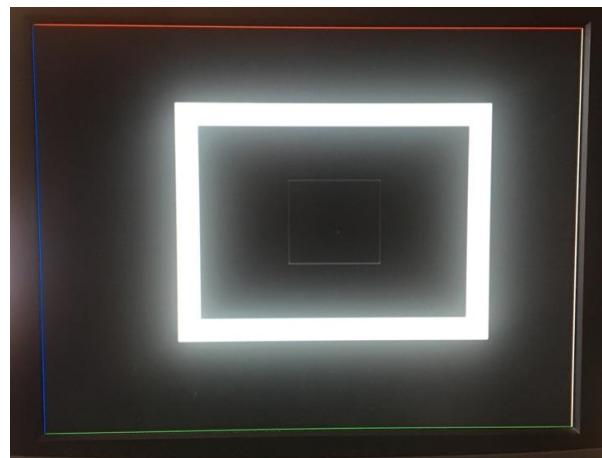


(b) Utilizando la FPGA.

Fig. 5.6: Resultado del filtro con el kernel 5 (enfoque) sobre Fig. 5.1.

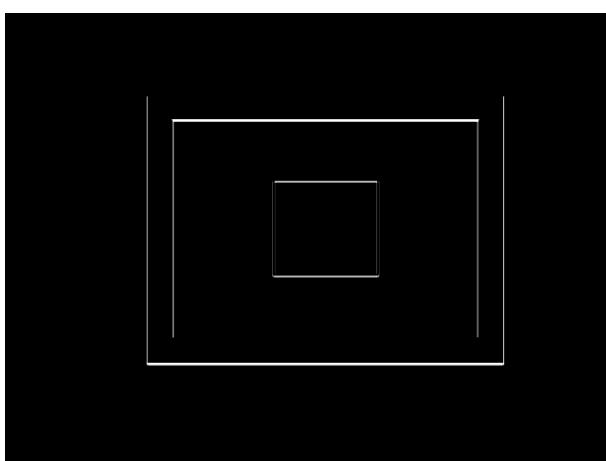


(a) Utilizando PaintShop.

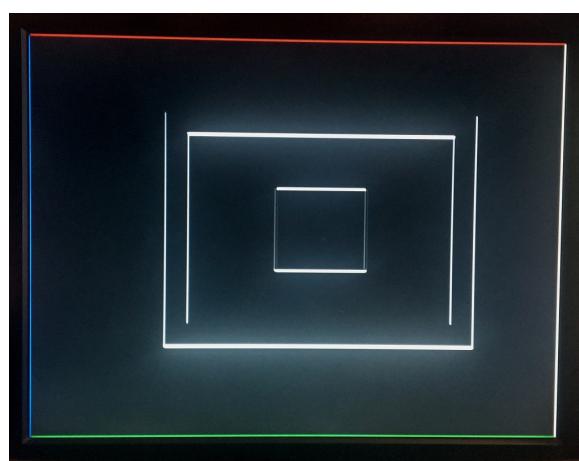


(b) Utilizando la FPGA.

Fig. 5.7: Resultado del filtro con el kernel 6 (suavizado) sobre Fig. 5.1.



(a) Utilizando PaintShop.



(b) Utilizando la FPGA.

Fig. 5.8: Resultado del filtro con el kernel 7 (filtro Norte) sobre Fig. 5.1.

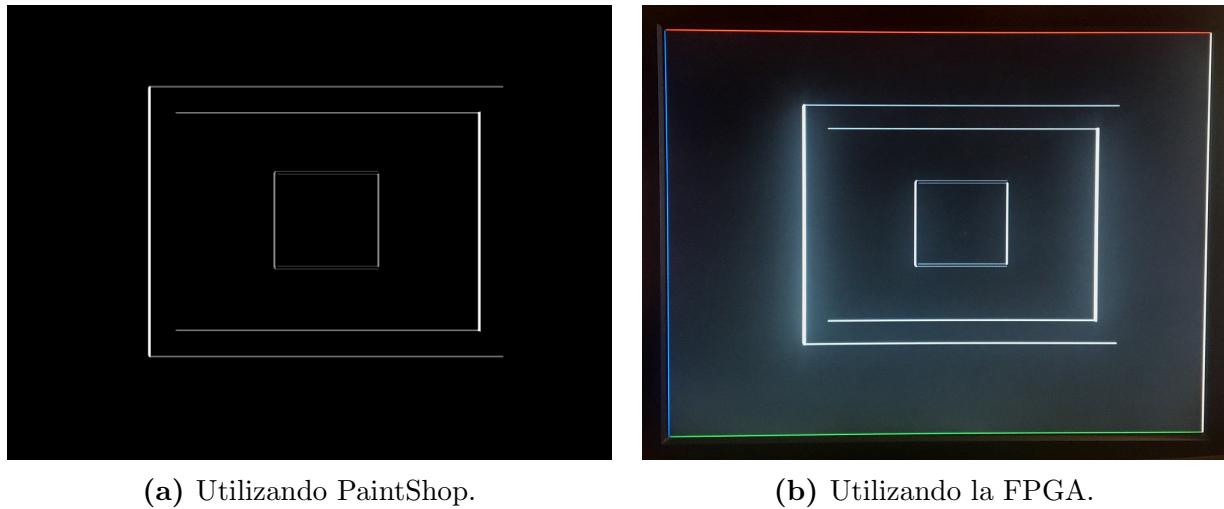


Fig. 5.9: Resultado del filtro con el kernel 8 (filtro Este) sobre Fig. 5.1.

En base a estos resultados se puede establecer que la implementación del sistema de filtrado es correcta. Además de ello, se ha comprobado que el sistema de captura de imagen también funciona correctamente y que el filtrado se realiza en tiempo real, se puede tomar una foto con el filtrado activo y se muestra el resultado directamente por pantalla. Si además se mantiene el *btnc* pulsado se toman fotos a 30 fps, es decir, se muestra la imagen de vídeo que el sensor captura y cómo se produce el filtrado en tiempo real, algo que sería computacionalmente muy costoso en software.

5.2. Test y cronogramas

El sistema funciona con un reloj de 100 MHz que entra a un divisor de frecuencia para crear también las señales de reloj de 50 y 25 MHz. El camino crítico del sistema es de 8,627 ns y se encuentra en la señal de *rdaddress* que conecta la salida del módulo de filtrado con la entrada de la memoria RAM de captura.

Un paso muy importante que se dio para solucionar los problemas que había en la comunicación entre el proceso de lectura de datos desde la RAM y el resultado del cálculo de la convolución fue el de utilizar esta imagen con colores puros (blanco y negro) e ir comparando los píxeles de la imagen de la simulación en Vivado y el PaintShop. Se trata de una zona crítica puesto que hay que interconectar dos sistemas que trabajan a diferentes frecuencias de reloj y se investigó usando el kernel 3 debido a que era el que más fallos producía.

Mediante la posibilidad de poder analizar el valor de cada píxel con PaintShop antes y después de la máscara de filtrado y que se permite observar también la posición exacta de los píxeles de entrada y salida en la convolución, se puede analizar el sistema con mucha precisión. En Fig. 5.10 se muestra la esquina superior izquierda de la imagen sin filtrar que se ha representado anteriormente (Fig. 5.1). Se ha resaltado el primer píxel distinto de 0x000000 (negro) que entra al sistema, en la parte inferior de la captura de pantalla recortada se observa

que la posición sobre la que se apunta con el cursor es (150, 86). La imagen está en RGB888, por ello el valor que muestra para cada color va de 0 a 255, en el caso que se trata dentro de la FPGA cada color varía entre 0 y 15, con lo que hay que hacer una reducción de la escala en cada color: $14/16 \approx 1 \rightarrow 0x111$.

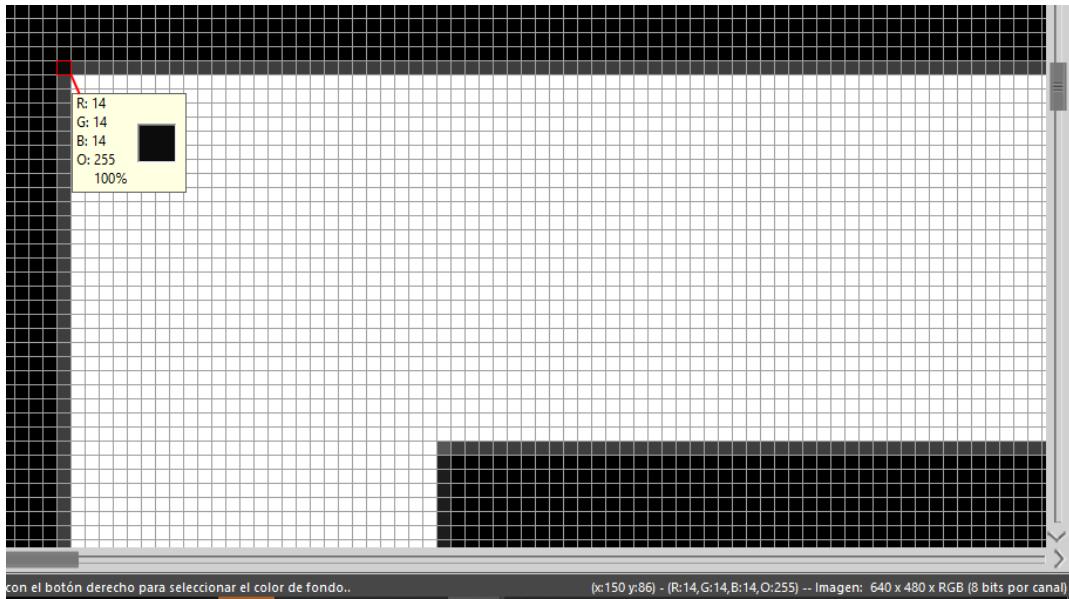


Fig. 5.10: Imagen ampliada de Fig. 5.1.

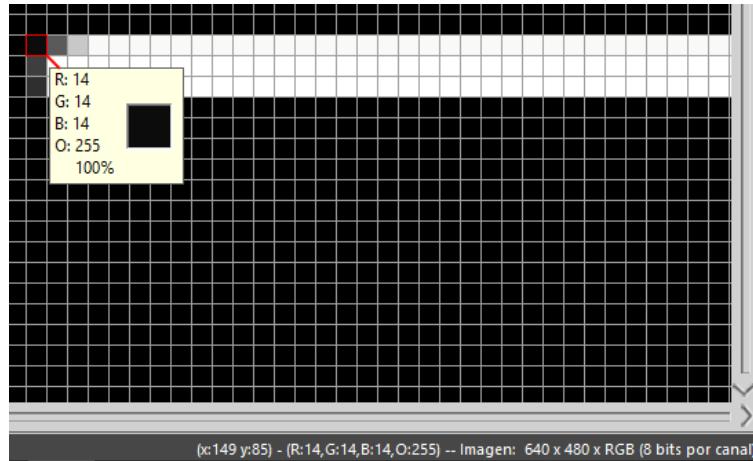


Fig. 5.11: Imagen ampliada de Fig. 5.1 tras el filtrado con el kernel 3.

Fig. 5.11 es el resultado que se obtiene con PaintShop tras filtrar con el kernel 3, máscara tipo ‘Sobel’ horizontal:

$$G = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

De esta forma, el resultado del filtrado proporciona el mismo valor de intensidad de color que antes de filtrar para la esquina superior izquierda, pues se multiplica por 1 en primer lugar (posición (3,3) de la matriz del kernel). Lo que si ocurre es que el píxel se ha desplazado de la posición (150, 86) a (149, 85), ya que el resultado de la convolución se guarda en la posición del valor del centro de la máscara.

En Vivado se obtiene el cronograma de Fig. 5.12, donde el primer píxel al que se está haciendo referencia llega cuando se lee la posición (150, 85), ya que se cargan, con ayuda de un contador (*cnt*), tres píxeles en un ciclo.

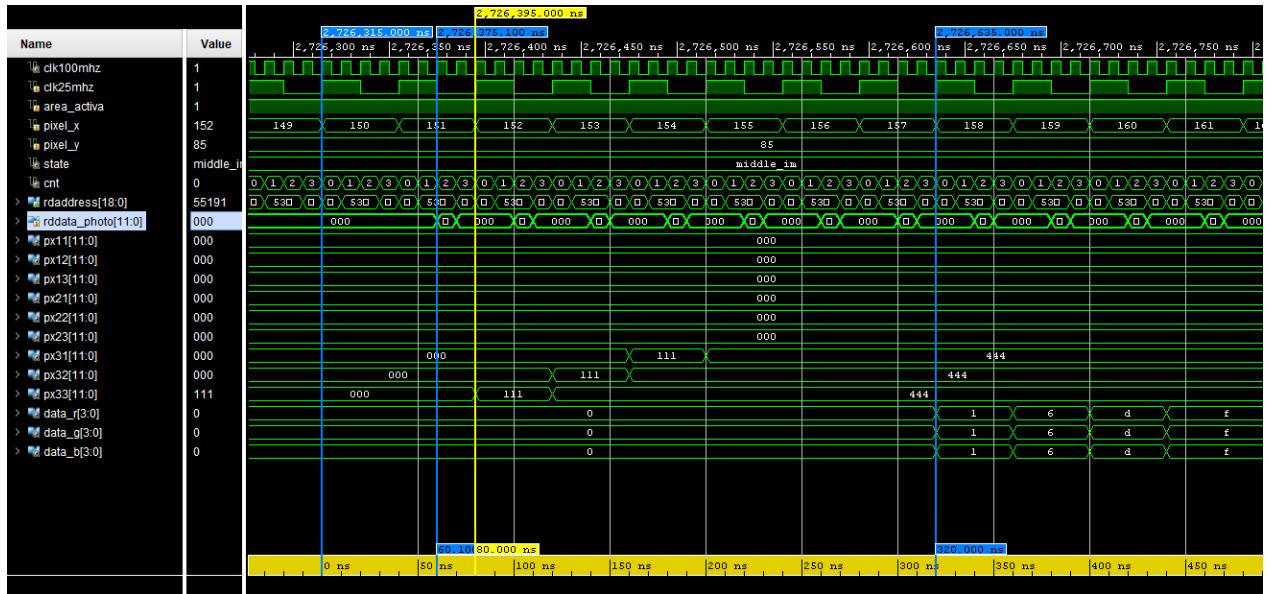


Fig. 5.12: Resultado en Vivado del proceso de filtrado de la esquina superior izquierda de Fig. 5.1.

La implementación de este contador funciona de forma sencilla, en cada ciclo de 25 MHz se cuenta hasta 4 a una velocidad de 100 MHz, permitiendo la carga desde memoria de los tres píxeles que se ilustran en Fig. 4.11. Cuando *cnt* = 0 se carga el píxel (1,3) de la máscara, cuando *cnt* = 1 se carga el píxel (2,3) y cuando *cnt* = 3 se carga el píxel (3,3). Pero hasta un ciclo siguiente no entran estos valores en la matriz, concretamente desde que *pixel_x* apunta a la posición 150 hasta que se tiene el dato de este, transcurren 60,1 ns y hasta los 80 ns después no entra a la matriz. Mediante los dos primeros marcadores en Fig. 5.12 se indica la diferencia de tiempos a la que se ha referido para el píxel 0x111.

Después de 320 ns se muestra por pantalla el resultado de la convolución (marcador de la derecha del cronograma), esto supone un retardo de 8 ciclos de reloj ($0,320 \cdot 25$) debido al cálculo del dato, pero sabiendo que el píxel se tiene que mostrar en (149, 85) este retardo aumenta hasta los 9 ciclos del reloj de 25 MHz.

Los datos de salida posteriores, 0x666, 0xdddd, 0xffff, corresponden a los cálculos que se muestran a continuación. Con ayuda de un programa creado en MATLAB y siguiendo el flujo de datos a través del DFG (Fig. 4.13) se van calculando rápidamente los valores deseados, de

no coincidir se repasa la zona del código VHDL donde se implementa la lógica que provoca ese fallo para comprobar lo que ocurre.

En hexadecimal y calculando cada color por separado:

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 4 \end{pmatrix} * \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} = 6,$$

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 4 & 4 \end{pmatrix} * \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} = d,$$

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 4 & 4 & 4 \end{pmatrix} * \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} = f.$$

A partir del valor de entrada 0x444, toda la primera fila tiene ese valor en la imagen original, luego la salida es 0xf para cada color mientras se mantengan esas entradas.

Este proceso de búsqueda de fallos se ha repetido hasta obtener los mismos resultados en la FPGA (simulación y demostración en pantalla) y en PaintShop Pro.

5.3. Recursos

Los resultados de los recursos de la FPGA que se han necesitado para la implementación del sistema propuesto se obtienen del resumen de utilización proporcionado por la simulación realizada mediante Vivado. Tabla 5.1 muestra un resumen general de todos los recursos utilizados en la FPGA en comparación con los disponibles.

Tabla 5.1: Tabla de utilización de recursos totales.

Resource	Used	Available	% Utilization
LUT	2633	63400	4,15
FF	1318	126800	1,09
BRAM	104	135	77,04
DSP48	6	240	2,50
IO	50	210	23,81
MMC	1	6	16,67

También se puede observar dicho resultado de forma gráfica en Fig. 5.13.

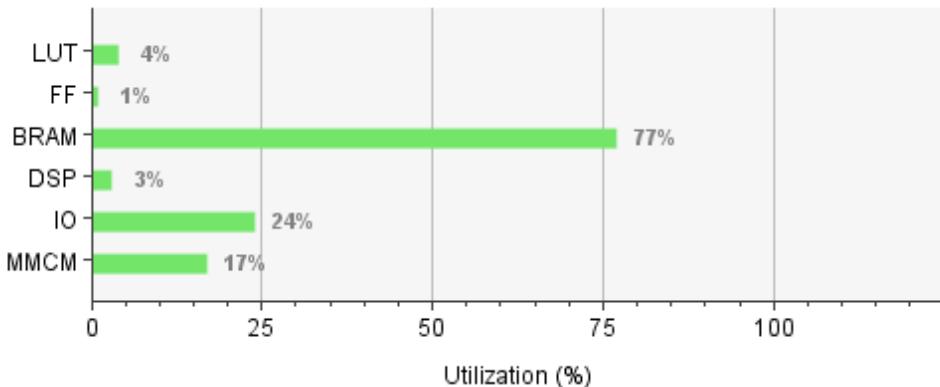


Fig. 5.13: Resumen de la utilización de recursos en la FPGA para el sistema completo.

Haciendo un análisis más detallado de los recursos utilizados, Tabla 5.2 recoge el reparto de recursos que se realiza de forma específica sobre los diferentes módulos implementados en el sistema completo.

Tabla 5.2: Tabla de utilización de recursos por módulo.

Módulos	LUT as Logic	FF	BRAM	DSP48	Slice
OV7670 capture	3	46	0	0	25
OV7670 control	54	91	0,5	0	25
SCCB sender	40	73	0	0	20
OV7670 register	14	18	0,5	0	6
Image refresh	7	25	0	0	11
Capture memory	505	18	103,5	0	228
Image filtering	1638	1140	0	6	601
VGA controller	424	68	0	0	185

La arquitectura que posee este sistema determina un elevado uso de memoria BRAM debido a que la foto que se captura mediante el sensor, es almacenada en su interior. Este almacenamiento no supone una grandísima cantidad de espacio de memoria (450 kB, Ec. 4.1) si se compara con los tamaños típicos de memoria que se pueden observar en el mercado para un ordenador de propósito general o un teléfono móvil de la actualidad. Pero la diferencia es que la memoria de la FPGA se encuentra embebida en un chip de silicio, lo que le permite una latencia de comunicación bastante inferior si se compara con la del ordenador o la del móvil. Se habla de que este sistema tiene una latencia de acceso a memoria de 2 ciclos de reloj, frente a los 9 o 20 ciclos de latencia de una memoria RAM comercial (eso es los mejores casos, con memorias RAM de gamas medias-altas).

Como también se puede apreciar en las tablas e imágenes anteriores, el núcleo del procesamiento matemático se encuentra en el módulo de filtrado de imagen, que es donde se realizan los cálculos de convolución. Este posee 6 módulos de DSP48 debido a la cantidad de cálculos que realiza en paralelo para implementar Ec. 3.2 en cada canal RGB. Puesto que el

cálculo se ha implementado junto con un pipeline, registrando todas las señales del cálculo de la convolución, este módulo también posee mayor cantidad de FF que el resto, 1140 en concreto.

5.4. Consumo de potencia

La herramienta Vivado Power Analysis proporciona un resumen muy completo e ilustrativo del consumo de potencia sobre la FPGA, este se incluye en Fig. 5.14. También indica sobre qué recursos se reparte el consumo indicando además de forma detallada la alimentación asignada a cada fuente de la FPGA, junto con su consumo de intensidad. Para hacer estos cálculos lleva a cabo serie de suposiciones iniciales, como por ejemplo la temperatura ambiente en 25 °C o la velocidad del flujo de aire en 250 LFM.

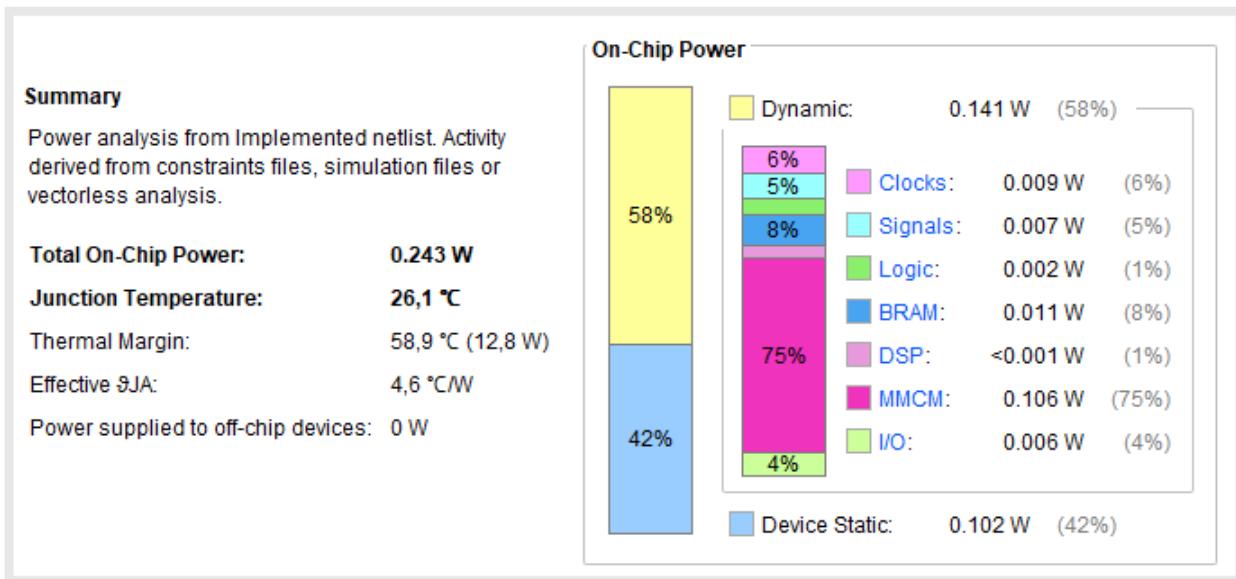


Fig. 5.14: Resumen del consumo de potencia en la FPGA.

Capítulo 6

Conclusiones

En resumen se puede establecer que se han alcanzado los objetivos planteados al comienzo del proyecto: se han conseguido implementar los distintos módulos propuestos inicialmente de forma eficiente para el filtrado de imágenes y se ha verificado satisfactoriamente su comportamiento.

Se ha realizado una implementación fiable del funcionamiento del controlador VGA, debido a que el controlador está muy bien depurado y libre de fallos. Esta interfaz ha resultado ser una herramienta muy útil para visualizar los resultados de las implementaciones de los filtros que se iban realizando y ha sido un módulo importante para la verificación del funcionamiento de la etapa de filtrado. En la memoria, se han explicado de forma sencilla los fundamentos teóricos de la interfaz VGA y del funcionamiento de las pantallas CRT.

Se han estudiado los distintos tipos de procesamiento de imagen y las diferencias entre los tipos de filtros que se pueden realizar. También se ha explicado el funcionamiento de los filtros de convolución y se ha logrado implementar en la FPGA un sistema de filtrado en tiempo real, mucho más eficiente en términos de rendimiento que un sistema de filtrado software.

Debido a la paralelización de los cálculos de la convolución este sistema es totalmente escalable a kernels de filtrado de mayor orden. Lo único que sería necesario es una mayor cantidad de recursos, sobretodo de almacenamiento que, como se ha observado durante el desarrollo del trabajo, se quedan justos para el almacenamiento de una imagen de resolución 640x480. Con el uso de FPGAs más grandes, las implementaciones hardware de estos algoritmos pueden llegar a tener mucha potencia.

Gracias al pipeline o paralelización temporal que se ha implementado el flujo de datos es calculado de forma más eficiente ya que permite un aumento importante en el throughput respecto a no implementarlo. El sistema podría ser aun más paralelo si se realizan cálculos en varios pipelines y se coordinan los flujos de datos para no sobreescribir los resultados.

Para completar el sistema también se ha realizado la implementación de los controladores de un sensor de imagen CMOS, con alto nivel de detalle. En un primer planteamiento del

problema se optó por utilizar una cámara web configurada de fábrica, de la que únicamente era necesario realizar la captación de los datos de la imagen a través de USB. Pero después se decidió que era mucho más experimental y desafiante tratar con un SoC en el que se integre un sensor CMOS, al que se puede acceder hasta una capa mucho más baja de abstracción para configurar un mayor número de parámetros.

En general, se puede concluir que se ha implementado un sistema de filtrado de imágenes sobre una FPGA, muy eficiente en consumo y cómputo de datos. Se trata de un diseño hardware con paralelizaciones temporales y espaciales, que cumple especificaciones en tiempo real. Esto se puede conseguir gracias al uso de FPGAs, las cuales son unos dispositivos muy adecuados y con muchas ventajas para la implementación de algoritmos de procesamiento de imagen, así como también lo son las GPUs.

Capítulo 7

Líneas futuras

Como líneas futuras, este Trabajo Fin de Máster sirve como una base de diseño sobre la que construir otros sistemas más complejos y eficientes en calidad de tiempo computacional o aplicabilidad.

El controlador VGA puede ser ampliado y mejorado para soportar otros tipos de resolución mayores. En este trabajo se trabajó con la extensión SVGA de 800x600 píxeles pero no se utilizó esa extensión ya que el sensor de imagen solo proporcionaba una resolución máxima en VGA. Además el hecho limitante de no disponer de mucha memoria de almacenamiento hizo desechar ese camino de trabajo. La implementación del controlador es fácilmente ampliable gracias a la modularidad de la librería de constantes de la que se dispone.

Si se disponen de más recursos de memoria se pueden llegar a realizar procesamientos a mucha mayor velocidad. Hay que recordar que la memoria RAM que se ha implementado solo dispone de un bus de salida de datos, por lo que si se utiliza mayor tamaño de almacenamiento se pueden dividir los datos y calcular mayor número de canales de salida en paralelo. Además, con las últimas tecnologías en el mercado de FPGAs se podría subir la frecuencia de reloj para aumentar el throughput del sistema. También destacar que las FPGAs de gamas superiores poseen mucha mayor capacidad de BRAM, como por ejemplo las Virtex UltraScale+ de Xilinx llegan a tener hasta 11,81 MB de memoria BRAM, 20 veces más que la FPGA de la que se dispone para este proyecto [37].

Otra opción en esta línea es la implementación de un sistema de almacenamiento de datos basado en memorias FIFO. Se podrían guardar desde la memoria RAM los datos de cada fila de la matriz de entrada, para que el cálculo de datos fuese más rápido. En ese caso se podría estudiar también la posibilidad de utilizar una memoria externa para guardar datos y que los que se guardan de la cámara vaya directamente a las FIFOs.

También se puede continuar el trabajo en el camino de adaptar el procesamiento de imagen para soportar otro tipo de técnicas de filtrado que requieren una computación más compleja [28]. Como por ejemplo, los filtros bilaterales, que son capaces de eliminar el ruido de una imagen de color gracias a la ejecución de técnicas estadísticas en base a los valores de los vecinos. O como también son interesantes los filtros inversos, los filtros de Wiener o

de Kalman [38], para mejorar la calidad de una imagen degradada por imperfecciones del sistema de adquisición o transmisión de imagen.

Por último, otra vertiente de trabajo futuro puede ser el de la utilización de este sistema como preprocesador para adaptar las imágenes para un algoritmo basado en aprendizaje máquina. Al sistema de este trabajo se le puede incluir una etapa posterior de segmentación y de reconocimiento de imagen utilizando redes neuronales (apartado 2.2) e incluso con procesamiento de vídeo, al cual este proyecto es fácilmente adaptable.

Apéndice A

Aspectos éticos, sociales, económicos y ambientales

En este apéndice se van a discutir algunos impactos relacionados con el proyecto. En particular se van a tratar los impactos éticos, sociales, económicos y ambientales.

A.1. Impactos ético-sociales

Muchas son las preocupaciones éticas relacionadas con el tratamiento de imagen en los diferentes sectores de aplicación. Su uso en medicina, uno de los que más se ha destacado en este proyecto, debe condicionarse en base al respeto de la dignidad, privacidad e intimidad del paciente.

En este sentido, existe un claro impacto relacionado con la publicación periodística y divulgación de estudios médicos, totalmente necesarias en este área del conocimiento para que se puedan permitir avances con mayor seguridad. Para ello, resulta necesario almacenar los resultados obtenidos de estudios o de casos reales en pacientes. He aquí donde interviene la fotografía médica digital, que permite plasmar tanto las causas como los resultados, de forma que se pueda tener en un futuro una base científica. El dilema ético se presenta en esta etapa, en el uso que reciben las posibles imágenes que se les toman a los pacientes.

Otro ejemplo es la aplicación de métodos de captación y procesamiento de imágenes en sistemas de asistencia a la conducción. Dichos vehículos permiten tomar imágenes de las personas que se encuentran en la calle. El problema ético en este sentido está relacionado con las identidades de dichas personas y la forma de actuar en consecuencia de las máquinas. De esta manera, no se trata solamente de un problema moral sobre el almacenamiento o transferencia de datos personales, sino que también se tiene que tener en cuenta qué debe hacer esa máquina, cuya cámara detecta a un peatón cruzándose por delante y que va a ser atropellado. ¿Qué se decide en este caso? ¿Y quién lo decide? ¿El programador? ¿La máquina? ¿La empresa que ha fabricado esa máquina?

Sin embargo, este mismo ejemplo puede verse desde el otro punto de vista, el vehículo puede haber detectado a la persona con suficiente tiempo como para desviar su trayectoria sin afectar a los pasajeros del automóvil. Visto así, este hecho ya si posee un carácter positivo para la sociedad, ya que sistemas como este pueden favorecer a la reducción de accidentes de tráfico y ayudar a las personas en la tarea de la conducción, evitando sus posibles errores.

En China, hoy en día las cámaras de vigilancia detectan a los ciudadanos en cuanto andan por la calle o se encuentran saliendo de un tren: identifican su rostro, analizan su fisonomía y son capaces de hasta adivinar la edad del sujeto con una certeza casi exacta. Todo este procesamiento tiene una serie de finalidades sociales muy positivas, la seguridad se encuentra presente en cualquier sitio de la ciudad en la que se haya implantado toda la infraestructura necesaria. Pero también hay que tener presente una serie de consecuencias que no tienen por qué ser deseadas por todos. Existe un control “innecesario” por parte de las empresas que llevan la supervisión de estos sistemas y que, por definición, tienen también sus propios intereses.

A.2. Impactos económicos

Actualmente, el procesamiento de imágenes es un atributo principal de la economía global. Con estos algoritmos se permite, por ejemplo, analizar el interés de un usuario a través de sus compras online o, simplemente, buscar un determinado producto utilizando una fotografía similar del mismo. De esta forma, los datos son procesados proporcionando a las empresas una visión más concreta en relación a las tendencias o hábitos de compra de los usuarios. Esto supone un incremento considerable en el acierto que se obtiene al ofrecerle al cliente un producto para que lo compre o simplemente mostrarle lo que con mayor probabilidad puede desear.

Por otra parte, continuando con el ejemplo expuesto de la asistencia autónoma a la conducción, cabe destacar que la industria del automóvil es uno de los sectores que más poder económico tiene en la actualidad, la cual desde los últimos años está apostando por este tipo de herramientas para los vehículos. La detección de carriles, personas, otros vehículos y señales de tráfico, son el resultado de procesos que requieren una elevada capacidad de cómputo. Para ello, sistemas como las FPGAs permiten salir al mercado, por ejemplo, un mes antes que el competidor más directo, esto provoca: mayor exaltación entre los usuarios, es decir, mayor número de ventas y lo que según mi punto de vista es más importante: **mayor capacidad para equivocarse antes y darse cuenta de dónde y por qué**.

A.3. Impactos ambientales

La implantación de sistemas de procesamiento de imágenes como el que se describe en este proyecto puede provocar una serie de consecuencias ambientales a tener en cuenta.

En el caso de las llamadas “*smart cities*”, el procesamiento, tratamiento y filtrado de

imágenes resulta de vital importancia para el correcto funcionamiento de determinados aspectos en la ciudad. Esto implica instalar diversos sistemas de captación de imágenes en lugares transitados por vehículos y peatones, lo que deriva en sistemas electrónicos repartidos por toda la ciudad y, por lo tanto, un aumento en el consumo de energía y una mayor contaminación electromagnética.

Esta contaminación medioambiental se debe principalmente a que, con la llegada de la tecnología 5G, cada vez se podrá mantener un mayor número de sistemas intercomunicados, lo que supone mayor cantidad de ondas de radio o comunicación similares. En este sentido, para los sistemas ADAS, el resto de usuarios necesitan la información de los vehículos de su alrededor, por ello ambos deben poder comunicarse e intercambiarse datos. También aquí resulta crucial la respuesta en tiempo real, algo que se puede lograr con mayor facilidad con diseños hardware.

Además, es necesario tener en cuenta que todos estos avances requieren un incremento en la producción, incremento que suele ser inversamente proporcional al buen uso y mantenimiento del medioambiente y sus recursos. De esta forma, los impactos que en esta sección se tienen en cuenta son vitales para que todo siga funcionando.

Apéndice B

Presupuesto económico

Recursos materiales				
Concepto	Unidades	Coste Unitario (€)	Amortización (%)	Coste Total (€)
Ordenador personal (licencias incluidas)	1	799,00	25	199,75
Nexys 4 DDR Digilent	1	282,00	25	70,50
Placa OV7670 VGA sensor CMOS 640x480	1	12,46		12,46
Cable conexión VGA	1	5,99		5,99
Pack de cables puente macho-hembra	1	3,99		3,99
Monitor de pantalla CRT	1	0,00		0,00
Encuadernación de memoria	1			40,00
TOTAL RECURSOS MATERIALES				332,69 €

Recursos humanos			
Nombre	Horas (h)	Coste/horas (€/h)	Coste Total (€)
Germán Bravo López	320	22,05	7.056,00 €
GASTOS GENERALES (costes ind.)		15 %	1.108,30 €
SUBTOTAL PRESUPUESTO			8.496,99 €
IVA		21 %	1.784,37 €
TOTAL PRESUPUESTO			10.281,39 €

El presupuesto total del proyecto asciende a diez mil doscientos ochenta y un euros con treinta y nueve céntimos

Apéndice C

Resultados sobre una imagen a color

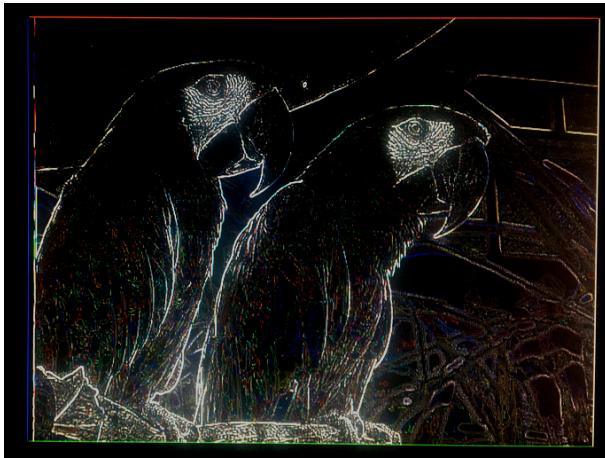
La imagen a color que se va a testear tanto en PaintShop Pro como en el filtrado de la FPGA es Fig. C.1. Es una fotografía obtenida de internet de resolución 640x480 píxeles.



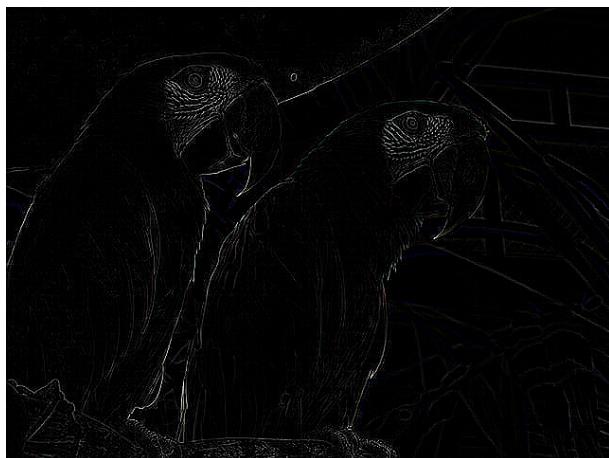
Fig. C.1: Imagen a color de resolución 640x480.



(a) Utilizando PaintShop.



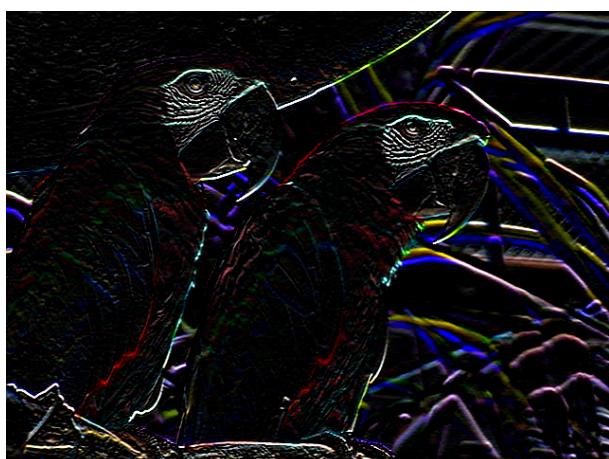
(b) Utilizando la FPGA.

Fig. C.2: Resultado del filtro con el kernel 1 sobre Fig. C.1.

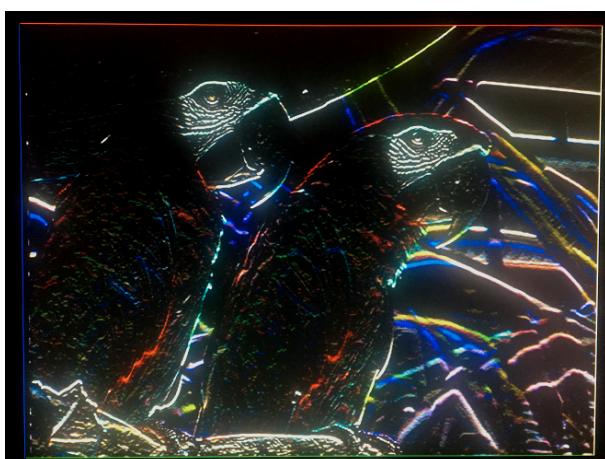
(a) Utilizando PaintShop.



(b) Utilizando la FPGA.

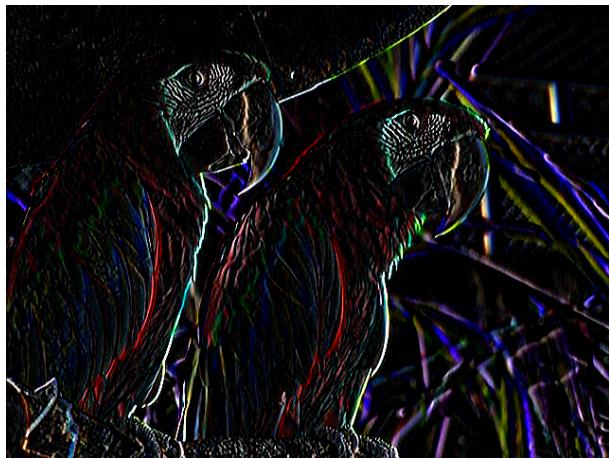
Fig. C.3: Resultado del filtro con el kernel 2 sobre Fig. C.1.

(a) Utilizando PaintShop.



(b) Utilizando la FPGA.

Fig. C.4: Resultado del filtro con el kernel 3 sobre Fig. C.1.



(a) Utilizando PaintShop.



(b) Utilizando la FPGA.

Fig. C.5: Resultado del filtro con el kernel 4 sobre Fig. C.1.



(a) Utilizando PaintShop.



(b) Utilizando la FPGA.

Fig. C.6: Resultado del filtro con el kernel 5 sobre Fig. C.1.

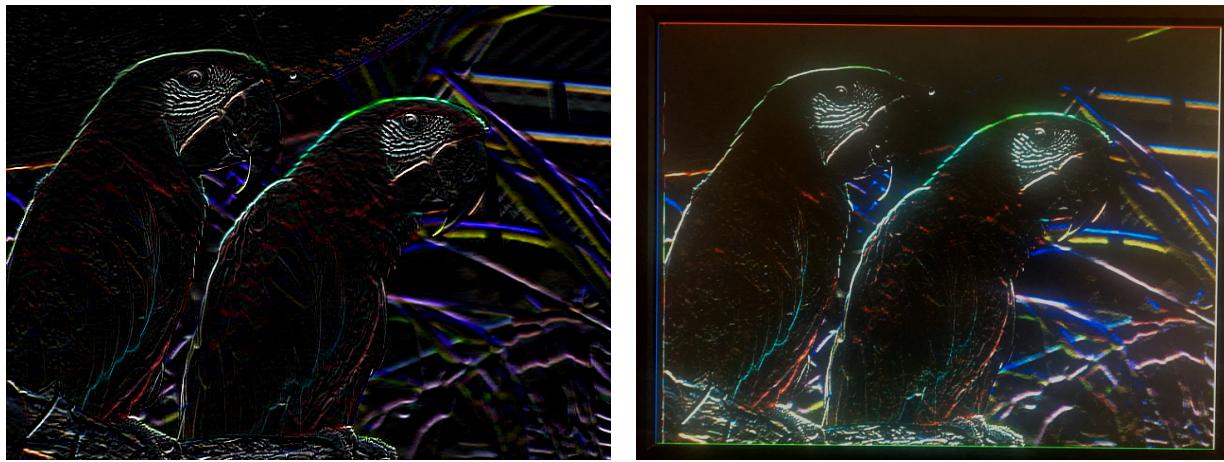


(a) Utilizando PaintShop.



(b) Utilizando la FPGA.

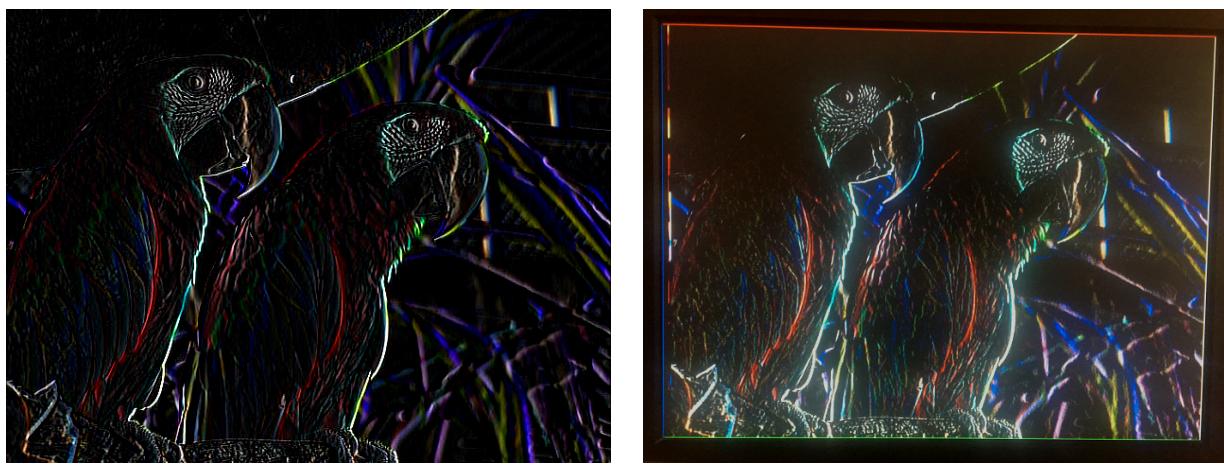
Fig. C.7: Resultado del filtro con el kernel 6 sobre Fig. C.1.



(a) Utilizando PaintShop.

(b) Utilizando la FPGA.

Fig. C.8: Resultado del filtro con el kernel 7 sobre Fig. C.1.



(a) Utilizando PaintShop.

(b) Utilizando la FPGA.

Fig. C.9: Resultado del filtro con el kernel 8 sobre Fig. C.1.

Bibliografía

- [1] J. C. Russ, *The Image Processing Handbook*, 6th ed. Boca Raton: CRC Press, April 2016.
- [2] B. Rabeh Amira, B. Faouzi, A. Hamid, and M. Bouaziz, “Computer-assisted diagnosis of alzheimer’s disease,” in *International Image Processing Applications and Systems Conference*. IEEE, 2014. [Online]. Available: <https://ieeexplore.ieee.org/>
- [3] M. Olfa and K. Nawres, “Ultrasound image denoising using a combination of bilateral filtering and stationary wavelet transform,” in *International Image Processing Applications and Systems Conference*. IEEE, 2014. [Online]. Available: <https://ieeexplore.ieee.org/>
- [4] F. J. Enríquez Aguilera, J. Martín Silva Aceves, S. Vianey Torres Argüelles, E. A. Martínez Gómez, and G. Bravo Martínez, “Utilización de gpu-cuda en el procesamiento digital de imágenes,” *CULC, UACJ*, 2018. [Online]. Available: <http://erevistas.uacj.mx/ojs/index.php/culcyt/article/view/2807/2566>
- [5] *Digilent Pmod™ Interface Specification*, Digilent® Inc., 1300 Henley Court, Pullman, WA 99163, November 2011. [Online]. Available: <https://www.digilentinc.com>
- [6] J. A. López Martín, “Introduction to digital systems and subsystems,” *Apuntes de Ingeniería de sistemas electrónicos analógicos y digitales*, 2019.
- [7] A. Al-Mahmood and M. Opoku Agyeman, “A study of fpga-based system-on-chip designs for real-time industrial application,” *International Journal of Computer Applications*, vol. 163, no. 6, 2017. [Online]. Available: <https://pdfs.semanticscholar.org/e642/6cafa835c457c7a49778768951a47a8dda14.pdf>
- [8] G. Oswaldo López Verástegui, “Implementación de algoritmos de procesamiento de imágenes en fpga,” Tesis de maestría en ciencias en ingeniería de cómputo con opción en sistemas digitales, Centro de Investigación en Computación del Instituto Politécnico Nacional, México, D.F., 2014.
- [9] P. P. Chu, *RTL HARDWARE DESIGN USING VHDL, Coding for Efficiency, Portability and Scalability*. Hoboken, New Jersey: John Wiley & Sons, Inc., 2006.
- [10] N. K. Ratha, “Computer vision algorithms on reconfigurable logic arrays,” Thesis, Michigan State University, 1996. [Online]. Available: <https://pdfs.semanticscholar.org/ae45/35cc52e676a9f28d669a2804b872cf854f3e.pdf>
- [11] “Field programmable gate array (fpga) market 2018 global size, share, development status, future trends, competitive landscape and industry expansion strategies 2023,” Market Watch, 2018. [Online]. Available: <https://www.marketwatch.com>

- [12] *7 Series FPGAs Data Sheet: Overview*, Xilinx®, Febrero 2018. [Online]. Available: <https://www.xilinx.com/products/silicon-devices/fpga.html>
- [13] *Presentación de la serie 7 de FPGAs*, Xilinx®. [Online]. Available: <https://slideplayer.com/slide/5967461/>
- [14] “Xilinx, official web page.” [Online]. Available: <https://www.xilinx.com>
- [15] M. Ramis, “Sistema de transmisión de imágenes bartlane,” Proyecto IDIS, Universidad de Buenos Aires. [Online]. Available: <http://proyectoidis.org/sistema-de-transmision-de-imagenes-bartlane/>
- [16] N. J. Loza Pérez and C. Rodriguez Doñate, “Optimización de una arquitectura en fpga para filtros gaussianos en imágenes,” Publicación en revista de divulgación, Universidad de Guanajuato, México, 2016. [Online]. Available: <http://www.jovenesenlacienca.ugto.mx/index.php/jovenesenlacienca/article/view/1264/888>
- [17] *Nexys4 DDR™ FPGA Board Reference Manual*, Digilent® Inc., 1300 Henley Court, Pullman, WA 99163, Abril 2016. [Online]. Available: <https://www.digilentinc.com>
- [18] *Artix-7 FPGA Product Brief*, Xilinx®, Inc., 2100 Logic Dr, San Jose, CA 95124, EEUU, 2018. [Online]. Available: <https://www.xilinx.com/support/documentation/product-briefs/artix7-product-brief.pdf>
- [19] *7 Series FPGAs Configurable Logic Block - User Guide*, Xilinx®, Septiembre 2016. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf
- [20] *7 Series FPGAs Memory Resources - User Guide*, Xilinx®, Febrero 2019. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf
- [21] *7 Series DSP48E1 Slice - User Guide*, Xilinx®, Marzo 2018. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf
- [22] E. R. Fossum, “Cmos image sensors: electronic camera-on-a-chip,” *IEEE Transactions on Electron Devices*, vol. 44, no. 10, pp. 1689–1698, Oct 1997. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/628824>
- [23] A. El Gamal and H. Eltoukhy, “Cmos image sensors,” *IEEE Circuits and Devices Magazine*, vol. 21, no. 3, pp. 6–20, May 2005. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/1438751>
- [24] *OV7670/OV7171 CMOS VGA (640x480) CAMERACHIP™ Sensor with OmniPixel® Technology*, OmniVision® Technologies Inc., 4275 Burton Dr, Santa Clara, CA 95054, EE.UU., Agosto 2006. [Online]. Available: http://web.mit.edu/6.111/www/f2016/tools/OV7670_2006.pdf
- [25] *OV7670/OV7171 CMOS VGA (640x480) CAMERACHIP™ Implementation Guide*, OmniVision® Technologies Inc., 4275 Burton Dr, Santa Clara, CA 95054, EE.UU., Septiembre 2005. [Online]. Available: <http://www.haoyuelectronics.com/Attachment/OV7670%20+%20AL422B%28FIFO%29%20Camera%20Module%28V2.0%29/OV7670%20Implementation%20Guide%20%28V1.0%29.pdf>

- [26] F. Giménez-Palomares, J. Monsoriu, and E. Alemany-Martínez, “Aplicación de la convolución de matrices al filtrado de imágenes,” *Modelling in Science Education and Learning*, vol. 9, no. 2, 2016. [Online]. Available: <https://polipapers.upv.es/index.php/MSEL/article/view/4524/4724>
- [27] R. J. Menéndez Alonso, B. M. López-Portilla Vigil, and M. E. Iglesias Martínez, “Diseño de un sistema de transferencia y procesamiento de imágenes sobre un fpga,” *Revista de la Facultad de Ingeniería U.C.V.*, vol. 28, no. 2, pp. 13–22, 2013. [Online]. Available: http://www.scielo.org.ve/scielo.php?script=sci_arttext&pid=S0798-40652013000200003
- [28] P. Kalra and S. Peleg, *Computer Vision, Graphics and Image Processing*, 1st ed. Madurai, India: Springer-Verlag Berlin Heidelberg, Diciembre 2006, vol. 4338.
- [29] John G. Proakis and Dimitris G. Manolakis, *Tratamiento digital de señales*, cuarta ed. Ribera del Loira, 28, 28042 Madrid: Pearson Prentice Hall, 2007. [Online]. Available: [https://www.academia.edu/35306793/Tratamiento_Digital_de_Se%C3%BEales_4_Ed._-John_G._Proakis_Dimitris_G._Manolakis](https://www.academia.edu/35306793/Tratamiento_Digital_de_Se%C3%BAales_4_Ed._-John_G._Proakis_Dimitris_G._Manolakis)
- [30] H. Ström, “A parallel fpga implementation of image convolution,” Master’s thesis, Linköping University, Linköping, Sweden, 2016. [Online]. Available: <https://liu.diva-portal.org/smash/get/diva2:930724/FULLTEXT01.pdf>
- [31] Página web. [Online]. Available: <https://www.fpga4student.com/>
- [32] A. K. Oudjida, M. L. Berrandjia, R. Tiar, A. Liacha, and K. Tahraoui, “Fpga implementation of i2c spi protocols: A comparative study,” in *2009 16th IEEE International Conference on Electronics, Circuits and Systems - (ICECS 2009)*, Dec 2009, pp. 507–510.
- [33] S. Cánovas Carrasco, “Sistema hardware de adquisición de vídeo basado en el sensor de imagen ov7670,” Master’s thesis, Universidad Politécnica de Cartagena, 2014.
- [34] J. C. Delgado Vázquez, M. A. García Martínez, R. Posada Gómez, and I. Herrera Aguilar, “Implementación de un sistema de adquisición de imágenes embebido en un fpga,” *Pistas Educativas*.
- [35] *Serial Camera Control Bus Functional Specification*, OmniVision® Technologies Inc., 4275 Burton Dr, Santa Clara, CA 95054, EE.UU., Marzo 2002. [Online]. Available: <http://www4.cs.umanitoba.ca/~jacky/Teaching/Courses/74.795-LocalVision/ReadingList/ov-sccb.pdf>
- [36] P. P. Chu, *FPGA PROTOTYPING BY VHDL EXAMPLES, Xilinx Spartan - 3 version*. Hoboken, New Jersey: John Wiley & Sons, Inc., 2008.
- [37] *Product tables and product selection guide*, UltraScale+ FPGA - Xilinx®, 2019. [Online]. Available: <https://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf>
- [38] “Mejora y restauración de imágenes,” *Documento de investigación*, Universidad de Valladolid. [Online]. Available: https://alojamientos.uva.es/guia_docente/uploads/2013/413/40833/1/Documento3.pdf