Desarrollo de un Sistema Embebido en FPGA para la detección de defectos en la fabricación de baldosas.

Reporte Nro. II. Julio 2019.
Autor: Tomás Medina, medinatomasariel@gmail.com

Resumen: *en este reporte se describe el proceso para poner utilizar y modificar una plataforma para procesamiento de video usando la placa Zybo Z7-20 y la cámara Pcam 5C. Además, se especifican los pasos para desarrollar (en lenguaje de alto nivel) un IP Core que realice procesamiento sobre las imágenes capturadas.*

1. Introducción

Una parte fundamental de este proyecto para la detección de fallas en baldosas es el la utilización de una plataforma en FPGA de procesamiento de video con la que se puedan realizar pruebas rápidamente, evitando tener que lidiar con el diseño de hardware. En este reporte se describen los pasos para utilizar y desarrollar sobre la plataforma propuesta.

El contenido de este reporte fue seguido utilizando el ambiente de Vivado Design Suite, siguiendo su workflow de desarrollo. Además, se realizó para ser usado en una placa Zybo Z7-20 (Figura 1) y con la cámara Pcam 5C (Figura 2), ambos fabricados por Digilent, por lo que su portabilidad puede estar limitada a estos componentes u otros muy similares del mismo fabricante.

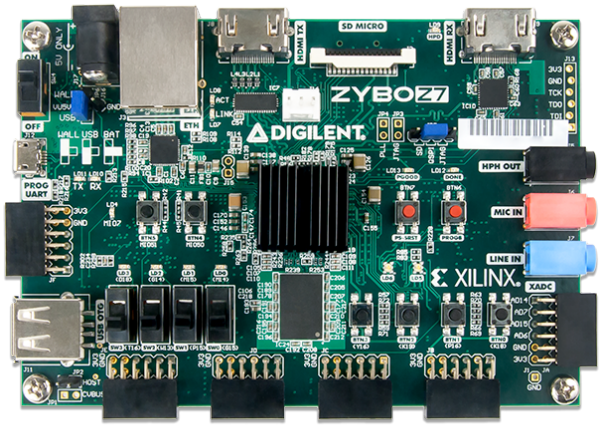

Fig. 1. Zybo Z7-20 de Digilent. Fuente: Digilent.

Fig. 2. Pcam 5C de Digilent. Fuente: Digilent.

Como primer paso, se busca configurar el diseño que irá en la FPGA para la captura de imágenes con la Pcam 5C y la salida de esa captura por el puerto HDMI de la Zybo. En esta primer etapa, se analizará el diagrama de bloques en Vivado, para así entender para qué sirven algunos de sus componentes y cómo podrían ser modificados para otros usos en en futuro. Como segundo paso, se explicará cómo subir a la placa el diseño realizado utilizando Vivado SDK, y cómo comunicarse con la placa en tiempo de ejecución mediante un puerto serial. En tercer lugar, se muestra y explica un código de HLS que aplicará un sobel en tiempo real al video capturado, se comentará cómo mejorar la latencia del resultado utilizando directivas y se explicará cómo probarlo y exportarlo e incluirlo en el diseño de la FPGA. Finalmente, se muestran los resultados de la plataforma en funcionamiento.

2. Configuración del Diagrama de bloques en Vivado

Vivado® es un software desarrollado por Xilinx para analizar, sintetizar, testear, implementar diseños de hardware a ser llevados a FPGAs. Se basa en la utilización de IP Cores (unidades de lógica reusable con propiedad intelectual) y diseños propios en algún lenguaje de descripción de hardware (ya sea VHDL o Verilog).

Para trabajar sobre él, Vivado posee un flujo de trabajo propio que consiste en: crear un proyecto utilizando el IDE; hacer el diseño de hardware (con un lenguaje de descripción de hardware o diagrama de bloques) y simularlo; sintetizar el diseño; implementar el diseño (ver cómo quedaría sobre la FPGA objetivo); realizar la simulación de tiempos; generar un *bitstream* para configurar la FPGA y subirlo a la placa para realizar las pruebas de funcionamiento en hardware. En la Figura 3 se puede ver gráficamente cómo es este *workflow*.

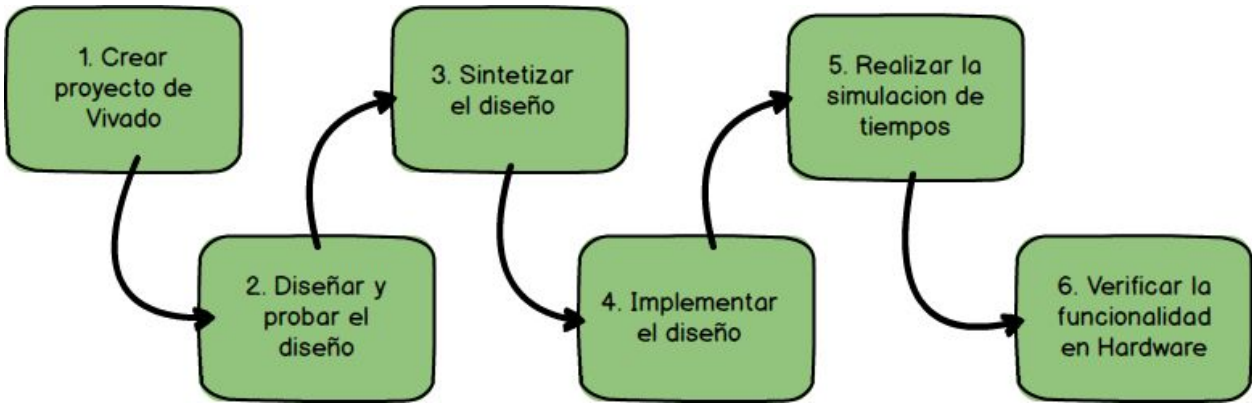

Fig. 3. Vivado Design Flow.

Para este proyecto en particular, se utiliza un proyecto para Vivado 2018.2 extraído de un repositorio de GitHub de Digilent, el cuál está específicamente diseñado para la placa objetivo (Zybo Z7-20) y la cámara a utilizar (Pcam 5C). Con este diseño se hace un pass-through de las imágenes capturadas con la cámara, pudiendo obtener la salida por el puerto HDMI Tx de la Zybo.

A continuación, se explicarán algunas de las partes más importantes del diagrama de bloques del proyecto.

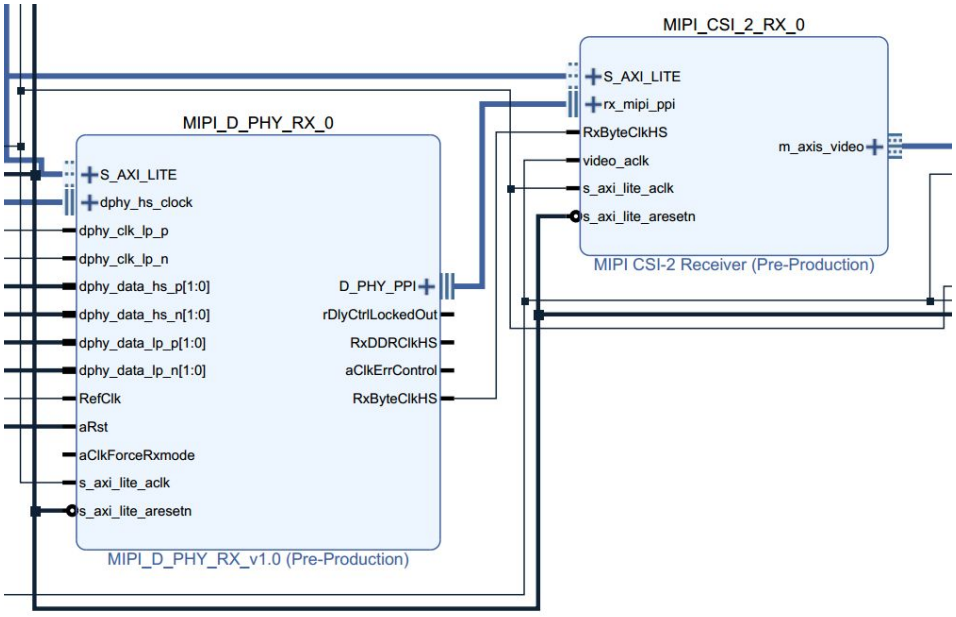

Fig. 4. IP Cores del diagrama de bloques correspondientes al manejo de la cámara.

En la Figura 4 se pueden ver los IP Cores que hacen la conversión de la señal proveniente de la cámara a un Stream de tipo AXI4. Un AXI4-Stream es un flujo de datos que sigue con un protocolo AXI. Provee una manera de comunicar dos bloques con un flujo de datos estable y de alta velocidad. Además, está de alguna manera estandarizado dentro del ambiente de desarrollo de Xilinx, por lo que utilizarlo provee una gran flexibilidad a la hora de implementar las interfaces para poder manipular los datos. En otras palabras, en este caso, que las imágenes capturadas por la Pcam se conviertan a un AXI4-Stream, hará más sencillo el posterior procesamiento de esas imágenes.

Algunos de las señales importantes de la interfaz Master de AXI son:

- *m_axis_video_tdata*: señal de salida la información transmitida en sí. En este caso es de 40 bits, pero puede variar.
- *m_axis_video_tlast*: señal de salida que indica que es el final del paquete enviado.
- *m_axis_video_tvalid*: señal de salida indica si el dato recibido actualmente es válido o no.
- *m_axis_video_tready*: señal de entrada por la que el AXI Slave indica que está listo para recibir la información.

Otras interfaces en los IP Cores de la Figura 4 son las interfaces AXI4-Lite que posee cada bloque. Estas serán utilizadas por el microprocesador para configurar los parámetros de la captura de imágenes en tiempo de ejecución. Tales parámetros pueden ser la resolución de

video, el *frame rate*, entre otros. *s_axi_lite_aclk* y *s_axi_lite_aresetn* corresponden a las señales de clock y reset de este AXI4-Lite. Otras señales, las que poseen el prefijo *dphy,* provienen desde fuera del diseño, están conectadas a ciertos pines de la placa por los que se obtendrá la información capturada con la cámara.

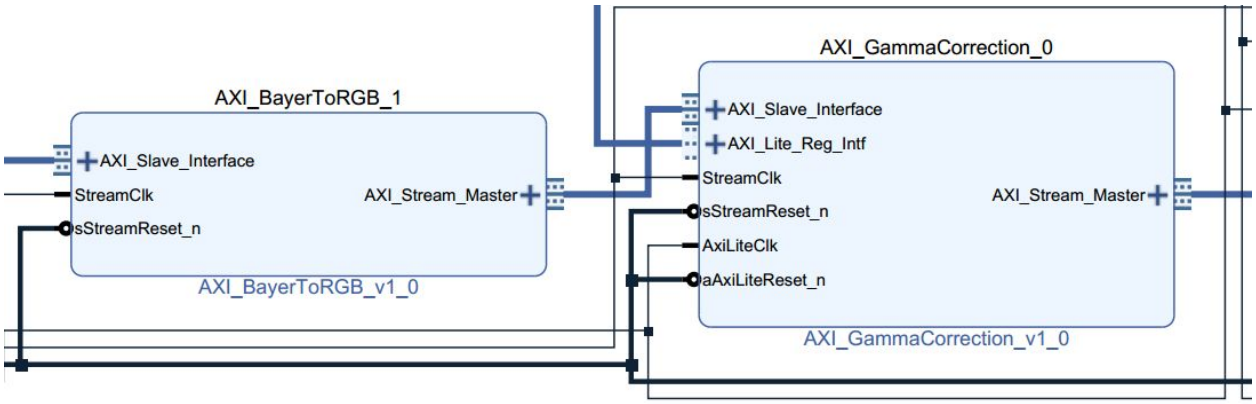

Fig. 5. IP Cores de procesamiento de imágenes AXI.

En la Figura 5 se pueden observar dos IP Cores que realizan operaciones sobre el AXI Stream saliente de los de la Figura 4. El primero, *BayerToRGB* realiza una conversión del formato crudo captado por la cámara a un espacio de color RGB. Un filtro, máscara o matriz de Bayer (Figura 6) es un tipo de matriz de filtros de colores rojo, verde y azul que se sitúa por arriba de los sensores digitales de las cámaras. Es un filtro que de alguna manera hace llegar la información de cada tipo de color al fotodiodo correspondiente en el sensor (información de la luz roja al fotodiodo que lee la intensidad del rojo y así). Este formato es útil en la captura de la imagen debido a la manera en que miden los sensores de imágenes, sin embargo sería complejo trabajar en ese formato para procesar la imagen digitalmente, razón por la que se hace la conversión a RGB.

La conversión se llama *interpolación cromática* o *bayer demosaicing*, y permite obtener una imagen de menor resolución que la original, en la que cada en píxel se calcula sus valores de colores a partir de los valores medidos en el formato del mosaico de bayer.

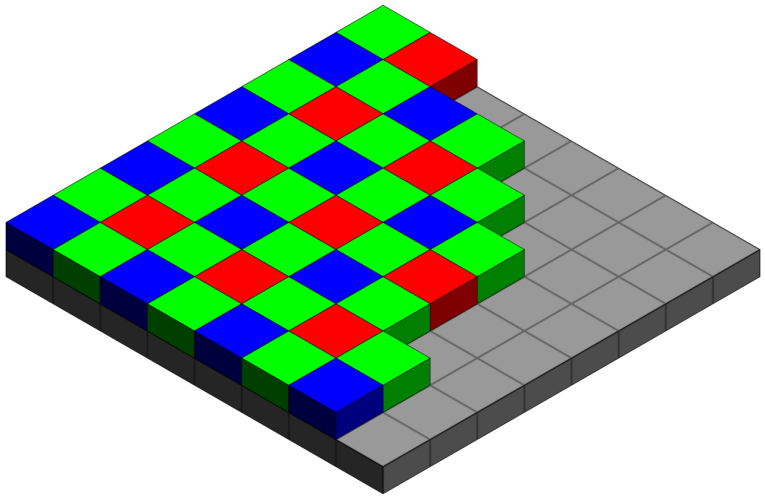

Fig. 6. Mosaico de Bayer. Fuente: Wikipedia.

Seguido de esto, está el IP Core responsable de la corrección gamma. La corrección gamma es una operación de procesado de imágenes que en ocasiones se usa para ajustar la luminancia y colores de manera que se perciban similarmente a cómo lo hacen los humanos. Explicado de otra manera, se aplica una corrección gamma para compensar la imagen como para que la visión humana pueda diferenciar y apreciar mejor los colores que en ella aparecen.

Una de las formas en la que se puede definir la corrección gamma es como en la Fórmula 1, siendo 255 la intensidad máxima de un píxel, I_{in} la intensidad del píxel de entrada, I_{out} la del píxel de salida. En la Figura 7 se pueden observar los resultados de la corrección para distintos valores de gamma.

$$I_{out} = 255 * \left(\frac{I_{in}}{255}\right)^{\gamma} \quad (1)$$

Fig. 7. Ejemplo de corrección gamma aplicada a una imagen, con distintos valores de gamma. Fuente: Wikipedia.

Siguiendo con el camino del AXI4-Stream que lleva las imágenes, en la Figura 8 se puede el siguiente bloque correspondiente a un AXI Video Direct Memory Access. Este IP Core se utiliza para tener una manera de comunicar (con un alto ancho de banda) la memoria con un AXI4-Stream de video. Es necesario debido a que muchas veces puede ser necesario manejar cambios en la frecuencia o en las dimensiones de la imagen. Además, este IP Core también provee una manera de sincronizar los frames de video salientes con una señal externa.

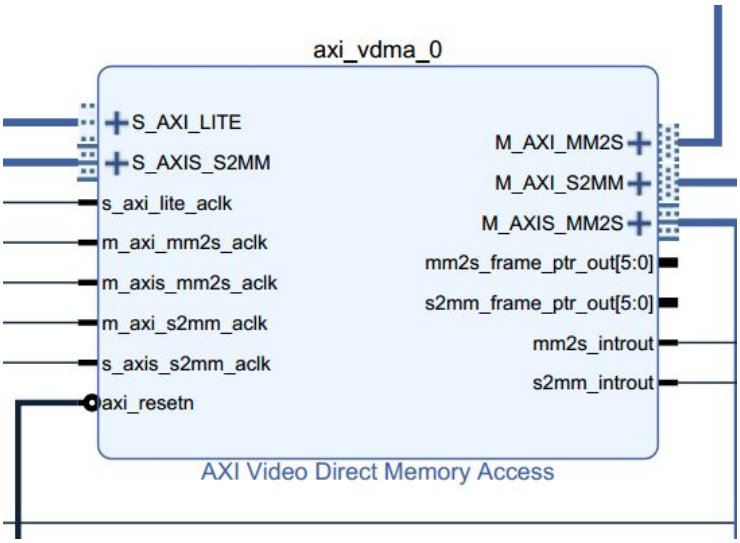

Fig. 8. IP Core correspondiente a AXI VDMA.

Como últimos pasos para que el video sea enviado a la salida, desde memoria, el IP Core de VDMA se comunica por su salida Master de AXI-Stream con el IP Core de AXI4-Stream to Video Out. Este último IP Core se puede ver en la Figura 9, y cuya principal función es la de convertir de un Stream AXI a una salida de video paralela con sus señales de sincronización. Utiliza también una fuente que define el *timing* por la interfaz *vtiming_in*, que en este caso puede variar para modificar el *frame rate* del video de salida (a 30 fps, 15 fps, 60 fps). Esta señal a *vtiming_in* viene desde un IP Core Video Timing Controller, el cual puede ser reconfigurado en tiempo de ejecución por el Zynq Processing System.

Ya habiendo convertido el AXI4-Stream a una señal de video “convencional”, queda convertirla para que sea transmisible por el puerto HDMI TX de la placa. Esto se hace mediante el uso del IP Core RGB to DVI Video Encoder.

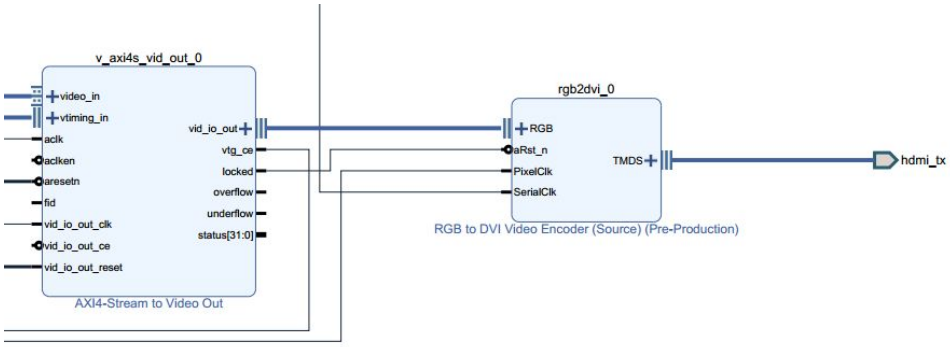

Fig. 9. IP Cores correspondientes a la conversión de AXI a rgb y rgb a DVI.

Por otro lado, está el bloque correspondiente al ZYNQ Processing System (Figura 10). Este corresponde al microprocesador presente en la placa. El mismo provee unas funcionalidades importantes al diseño. En primer lugar, es la fuente de gran parte de los clocks utilizados por otros bloques (*FCLK_CLK0*), así como también del reset (*FCLK_RESET0_N*). En segundo lugar, es donde correrá el programa, que permitirá configurar algunas de las opciones de salida tales como el *frame rate*, la resolución de salida, la corrección gamma, entre otros. Estas configuraciones se harán mediante M_AXI_GP0, la cual es una salida de tipo AXI4-Lite que va hacia un AXI Interconnect.

El AXI Interconnect (Figura 11) funciona, como bien su nombre lo indica, para interconectar interfaces AXI. En este caso particular, sirve para conectar un único Master AXI4-Lite del procesador con múltiples AXI4-Lite Slaves (AXI VDMA, Clocking Wizard, Video Timing Controller, MIPI_D_PHY_RX, MIPI CSI-2 Receiver, Gamma Correction). De esta manera, el procesador puede actuar para configurarlos a todos mediante una sola salida AXI.

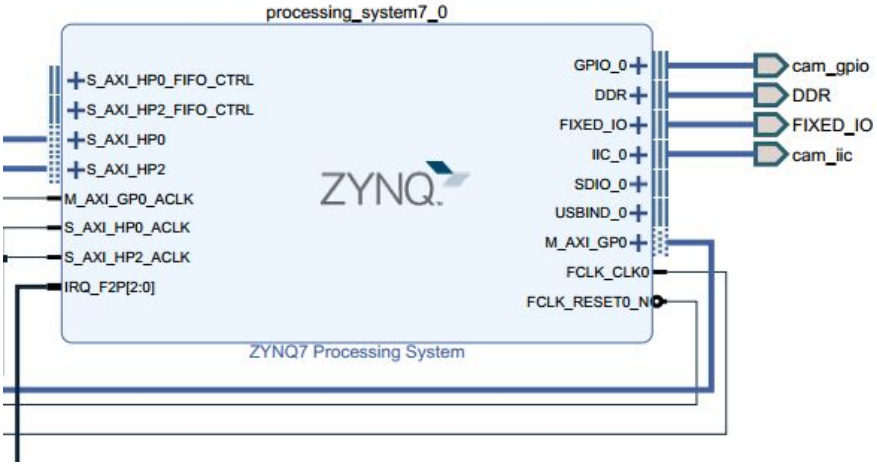

Fig. 10. IP Core de ZYNQ Processing System.

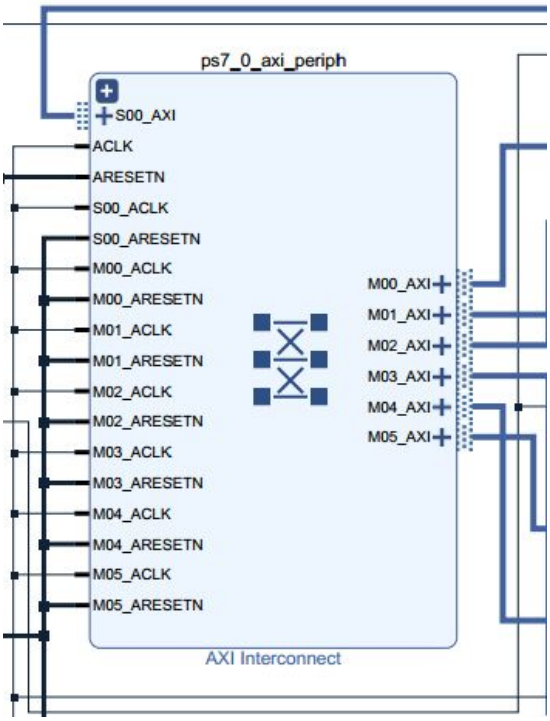

Fig. 11. IP Core de GPIO.

Tras haber definido el diagrama de bloques en Vivado, el próximo paso es sintetizarlo, implementarlo y generar el bitstream. En la síntesis, Vivado convierte el diseño RTL a una representación en puertas lógicas, tratando de optimizar de forma balanceada el uso de recursos y la performance. Si bien se pueden seleccionar ciertas directivas para indicar si se quiere minimizar el uso de recursos o la performance resultante, en este caso se utilizan las opciones estándar de síntesis.

Luego de la síntesis sigue la implementación. En ella se toma la representación en puertas lógicas resultante y se adapta esa solución a la placa objetivo (en este caso, la XC7Z020-1CLG400C). Se ubican partes del diseño en la FPGA, realizando optimizaciones para minimizar el área o maximizar la performance. Al final de esta etapa, se obtiene un reporte de la utilización de la placa, sirviendo como indicador de cuánto más se le puede agregar en el futuro a este diseño sin agotar los recursos de la placa.

Por último, se genera el bitstream, que es el código que, al ser enviado a la placa, programa la FPGA con el diseño obtenido de la implementación. Luego de esto, se exporta el hardware incluyendo el bitstream para trabajar con él en el SDK.

3. Uso de SDK y subida del programa a la placa

Xilinx SDK es un IDE para el desarrollo sobre los diseños de hardware de Vivado. Esta herramienta sirve para desarrollar el software que correrá en el microprocesador de la Zybo Z7-20.

En el presente caso, el software a subir permitirá a la placa comunicarse con la PC para recibir las configuraciones deseadas, y configurar en tiempo de ejecución los parámetros de los Cores IP del hardware. Por ejemplo, hará posible cambiar en tiempo de ejecución la resolución de salida.

El primer paso consiste en la creación de un proyecto para una nueva aplicación. En sus opciones se indica que se cree un nuevo BSP (Board Support Package) con el hardware anteriormente exportado. Un BSP es una capa de software que provee los drivers requeridos para que se pueda manejar el hardware. Pone a disposición, por ejemplo, las direcciones de memoria en las que escribir para configurar algunos parámetros del hardware.

Siguiendo, se selecciona crear el proyecto en C++ con una aplicación vacía. Se importan los archivos correspondientes al *main* y a los drivers *hdmi* (para configurar la resolución de salida y el *frame rate*) y *ov5640* (para configuraciones del sensor de la cámara). En el *main* se define el programa principal que se encarga de comunicarse con la PC para recibir las configuraciones, y con los drivers para hacer esos cambios en las configuraciones.

A continuación se conecta la cámara a la placa como se muestra en la Figura 12. Luego, se conecta la placa a la PC mediante USB. Se selecciona la opción del SDK para programar la FPGA. Cuando el led de status de la Zybo esté en verde, ya estará programada la lógica de hardware.

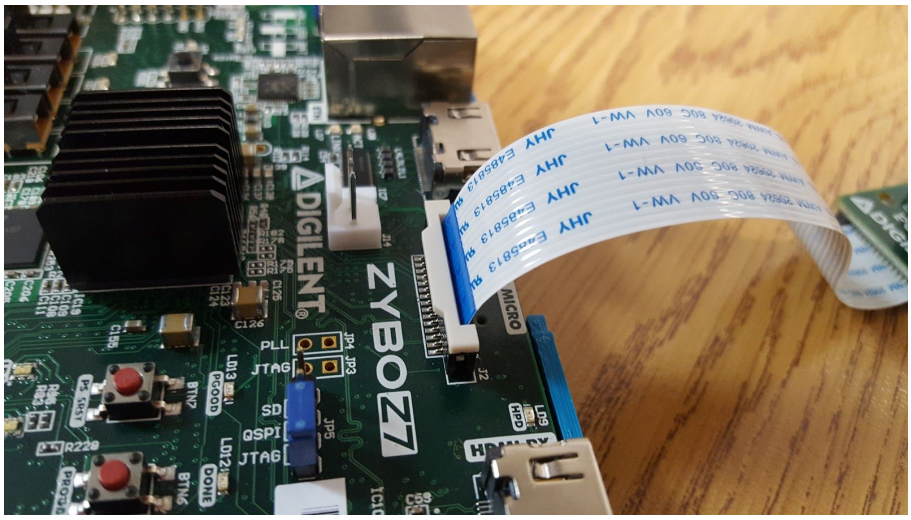

Fig. 12. Conexión de la cámara Pcam 5C a la Zybo.

Por último, se abre una terminal para la comunicación con un puerto serie. Se configura con los parámetros como en la Figura 13. Mediante esta terminal se podrá realizar la

comunicación con la placa en tiempo de ejecución. Se selecciona en el SDK la opción de correr el software, y ya debería estar funcionando el pass-through de video. Para ver el resultado es necesario conectar a un monitor por el puerto HDMI Tx de la placa.

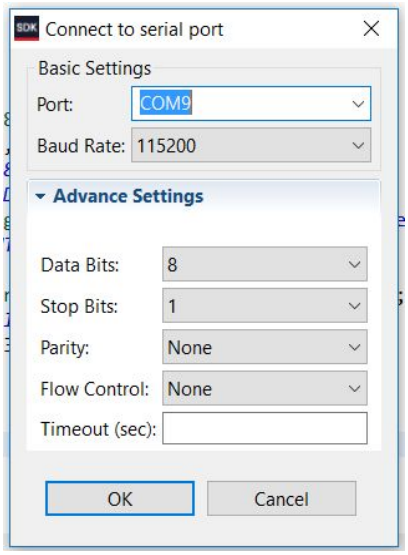

Fig. 13. Parámetros para la comunicación serial con la placa.

4. Desarrollo de un IP Core de procesamiento de video en HLS

Vivado HLS (High Level Synthesis) es un componente del ambiente de Xilinx que permite acelerar drásticamente el desarrollo de IP Cores mediante la posibilidad de definir los algoritmos requeridos en lenguajes de programación de más alto nivel que los lenguajes de descripción de hardware como VHDL o Verilog. De esta manera, los algoritmos se pueden desarrollar en C o C++ de forma ágil y probarse fácilmente. Además, Xilinx provee soporte para incluir interfaces tales como AXI4, AXI4-Stream, FIFO. Por otro lado, también provee la biblioteca xfOpenCV (o su equivalente anterior, HLS Video Library), que contiene gran cantidad de operaciones de procesado de imágenes similares a las de OpenCV, las cuales además están optimizadas y son sintetizables en FPGA. Estas últimas serán de especial utilidad para el procesado de las imágenes capturadas.

En esta sección se hará uso del programa Vivado HLS 2018.2 junto con la biblioteca HLS Video Library para implementar y testear un IP Core de un filtro sobel para la detección de bordes en imágenes, y más tarde incluirlo en el diseño descrito en los anteriores apartados. El código y su explicación fueron tomados de un tutorial.

El operador sobel es un operador diferencial que se utiliza en procesamiento de imágenes para la detección de bordes. Se basa en la detección de los cambios de intensidad de los píxeles en su vecindad. Para calcular el gradiente se utiliza una matriz de convolución que permite, al ir iterando, multiplicar los valores de cada píxel de la imagen original uno a uno con los valores de la matriz para obtener cada píxel de la imagen resultante. En la Figura 14 se muestra un ejemplo del cálculo de un píxel de la imagen de salida. En este ejemplo, se calcula

multiplicando la primera celda de la subimagen de la imagen original por la primera celda de la matriz de convolución, sumado a la segunda celda de la subimagen por la segunda celda de la matriz, y así.

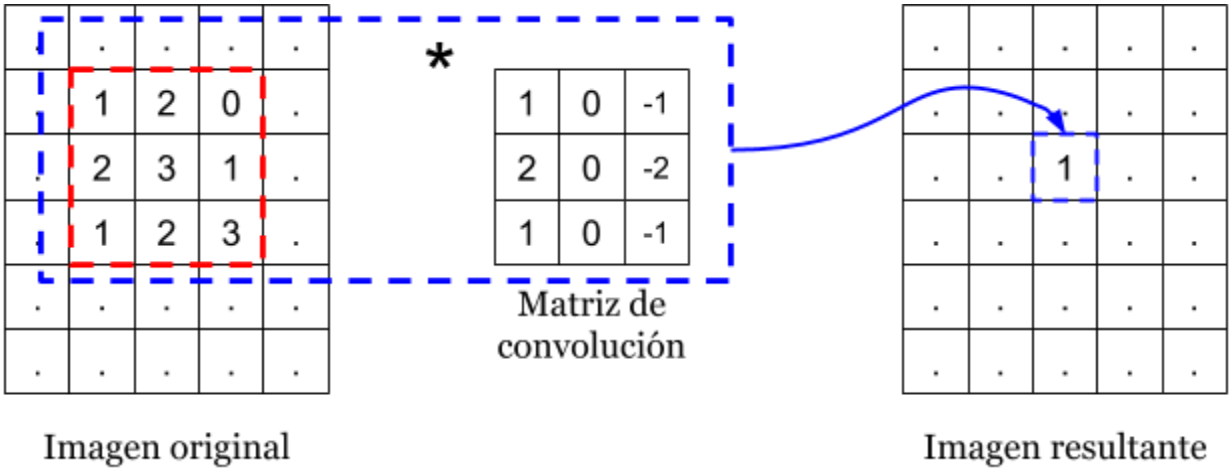

Fig. 14. Ejemplo de multiplicado por matriz de convolución.

Se puede calcular el gradiente X (los bordes verticales, como en el ejemplo), el gradiente Y (los bordes horizontales), o, como en este caso, se calculan ambos para luego "sumarlos" en una única imagen resultante (con los bordes en ambas direcciones).

El pipeline para calcular el sobel de esta última manera consiste en: pasar la imagen a escala de grises; aplicar un desenfoque gaussiano; aplicar sobel en ambas direcciones resultando en dos imágenes; "sumar" los valores de cada imagen en una sola; pasar la imagen a RGB (aunque el resultado sea en escala de grises en realidad). Gráficamente, este pipeline de procesamiento se puede ver en la Figura 15.

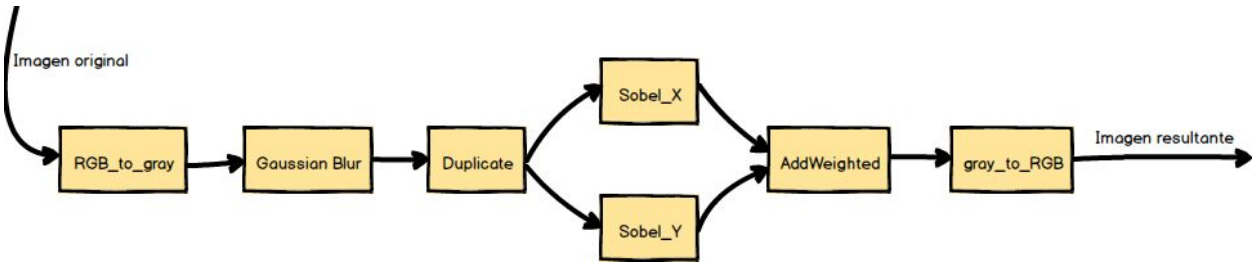

Fig. 15. Pipeline de procesamiento del sobel.

En cuanto a código, la función que hace este sobel se define de la siguiente manera:

```
void sobel_hls(AXI_STREAM& INPUT_STREAM, AXI_STREAM& OUTPUT_STREAM)
{
	#pragma HLS INTERFACE axis port=INPUT_STREAM
	#pragma HLS INTERFACE axis port=OUTPUT_STREAM
	RGB_IMAGE  img_0(MAX_HEIGHT, MAX_WIDTH);
	GRAY_IMAGE img_1(MAX_HEIGHT, MAX_WIDTH);
	GRAY_IMAGE  img_2(MAX_HEIGHT, MAX_WIDTH);
	GRAY_IMAGE  img_2a(MAX_HEIGHT, MAX_WIDTH);
	GRAY_IMAGE  img_2b(MAX_HEIGHT, MAX_WIDTH);
	GRAY_IMAGE  img_3(MAX_HEIGHT, MAX_WIDTH);
	GRAY_IMAGE  img_4(MAX_HEIGHT, MAX_WIDTH);
	GRAY_IMAGE  img_5(MAX_HEIGHT, MAX_WIDTH);
	RGB_IMAGE  img_6(MAX_HEIGHT, MAX_WIDTH);
	;
	#pragma HLS dataflow
	hls::AXIvideo2Mat(INPUT_STREAM, img_0);             // 1
	hls::CvtColor<HLS_BGR2GRAY>(img_0, img_1);          // 2
	hls::GaussianBlur<3,3>(img_1,img_2);                // 3
	hls::Duplicate(img_2,img_2a,img_2b);                // 4
	hls::sobel<1,0,3>(img_2a, img_3);                   // 5
	hls::sobel<0,1,3>(img_2b, img_4);                   // 6
	hls::AddWeighted(img_4,0.5,img_3,0.5,0.0,img_5);    // 7
	hls::CvtColor<HLS_GRAY2RGB>(img_5, img_6);          // 8
	hls::Mat2AXIvideo(img_6, OUTPUT_STREAM);            // 9
}
```

Como se puede observar en el código, la función recibe un input AXI4-Stream y retorna un output AXI4-Stream. Retomando las razones expuestas en secciones anteriores, esto le dará portabilidad al IP Core resultante ya que las interfaces AXI son un estándar en el diseño de FPGAs. Las instrucciones marcadas con los comentarios 1 y 9 corresponden a las conversiones desde AXI a Mat (la clase contenedora de imágenes por excelencia) y viceversa respectivamente. El resto de las instrucciones comentadas (del 2 al 8) corresponden a cada una de las funciones del pipeline antes descrito.

Por otro lado, los pragmas *HLS interface* se utilizan para especificar cómo deben crearse los puertos RTL en la síntesis de puertos. En este caso, se especifica que estos puertos deben ser de tipo AXI4-Stream.

Por último, cabe resaltar que se utiliza el pragma *HLS dataflow*. Esta directiva indica que las funciones subsecuentes pueden superponerse durante su ejecución, lo que permite aumentar la concurrencia de la implementación e incrementar el *throughput* del diseño. En la Figura 16 Se pueden ver dos gráficos temporales, el primero (A) correspondiente a una función sin la utilización de este pragma y el segundo (B) utilizándolo. Claramente, se observa una mejoría en el rendimiento en el caso B respecto del A.

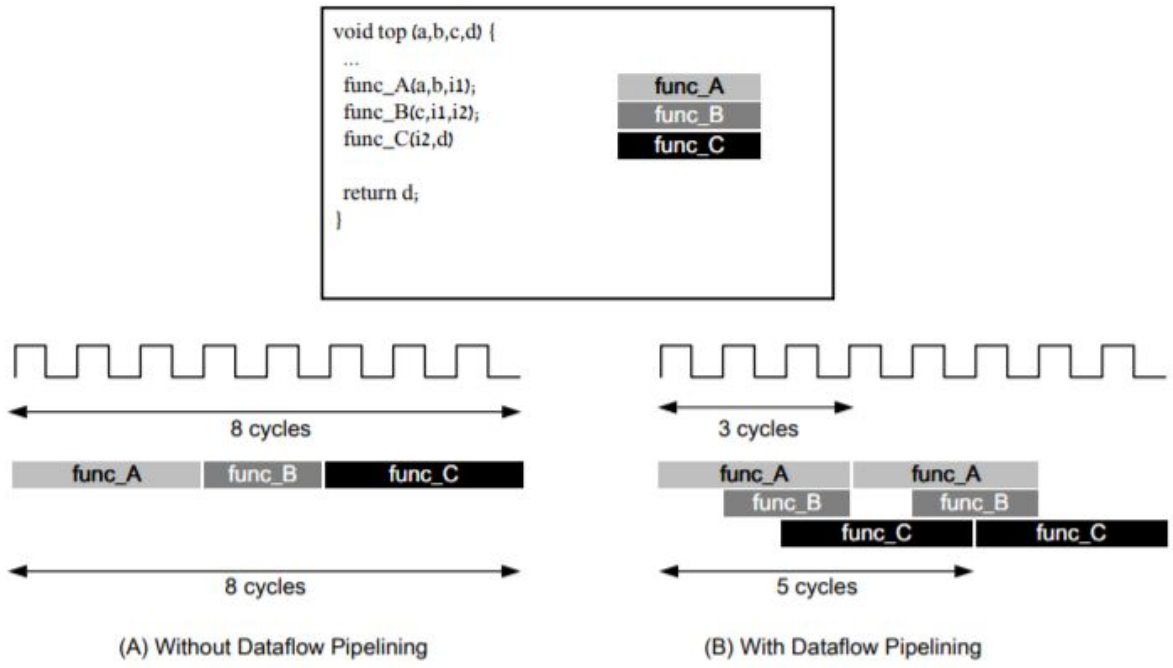

Fig. 16. Comparación de una función con y sin el uso del pragma dataflow. Fuente: Xilinx.

Ya habiendo explicado la función implementada, el siguiente paso es verificar que funcione como se espera. Para ello, se hace una simulación en C, donde se simula en software los resultados que arrojaría el hardware. Se requiere un código de testing (un testbench), como el que se muestra a continuación:

```
int main (int argc, char** argv)
{
        IplImage* src;
        IplImage* dst;
        AXI_STREAM src_axi, dst_axi;
        src = cvLoadImage("test.bmp");                                  //1
        dst = cvCreateImage(cvGetSize(src), src->depth, src->nChannels);
        IplImage2AXIvideo(src, src_axi);                                //2
        sobel_hls(src_axi, dst_axi);//src->height,src->width);          //3
        AXIvideo2IplImage(dst_axi, dst);                                //4
        cvSaveImage("op.bmp", dst);                                     //5
        cvReleaseImage(&src);
        cvReleaseImage(&dst);
}
```

Como se puede apreciar, en las instrucciones marcadas con 1 y 5 se realizan la carga y guardado de imágenes desde archivos con funciones de la biblioteca OpenCV. Las instrucciones 2 y 4 hacer las conversiones entre las imágenes de OpenCV a un AXI4-Stream (ya que la función

implementada tiene como interfaces AXI4-Streams). Por último, la instrucción 3 hace el llamado a la función definida anteriormente.

Tras ejecutar la simulación en C, se puede ver la imagen resultante para verificar que el funcionamiento es el deseado.

El siguiente paso consiste en sintetizar el diseño. Para ello es importante especificar la placa objetivo para obtener los reportes de utilización adecuados. Tras sintetizarlo, se pueden visualizar los siguientes reportes:

- Reporte de tiempos (*performance estimates*): indica cuál es el tiempo de clock necesario estimado, la latencia en clocks de la función sintetizada, entre otros detalles.
- Reporte de utilización (*utilization estimates*): indica la cantidad (y porcentajes) de recursos de la FPGA (BRAM, DSPs, FFs, LUTs) que se supone que se van a ocupar en la implementación del diseño.
- Interfaces: especifica las interfaces que tendrá el IP Core.

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	10.049	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
2087192	2104938	2087181	2104922	dataflow

Detail

Fig. 17. Reporte de tiempos.

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	34
FIFO	0	-	107	454
Instance	9	39	5893	11292
Memory	-	-	-	-
Multiplexer	-	-	-	36
Register	-	-	6	-
Total	9	39	6006	11816
Available	280	220	106400	53200
Utilization (%)	3	17	5	22

Fig. 18. Reporte de utilización.

En las Figuras 17 y 18 se pueden visualizar los reportes obtenidos. De estos reportes se puede destacar que lo que más se utilizará son las LUTs con un 22% de las LUTs de la placa necesarias, que se puede sumar al porcentaje de utilización obtenido en la implementación del *pass-through* para verificar que este IP Core tenga posibilidades de entrar. En este caso, debido a los recursos de la placa objetivo, se sabe que tiene lugar de sobra para incluir el sobel.

Otro resultado importante arrojado por la síntesis es la representación gráfica del schedule de la función. En ella se puede observar cómo se comunican, cuanto tardan y cómo es la concurrencia de la solución con las directivas planteadas. En la Figura 19 se puede ver el schedule de esta solución. Como se puede ver en la etapa 9, gracias a la directiva dataflow especificada, los dos sobel (gradientes X e Y) se calculan en simultáneo, minimizando así la latencia.

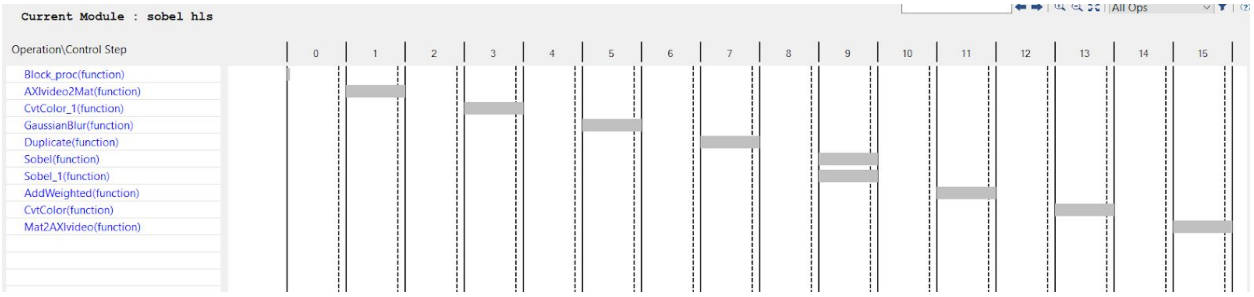

Fig. 19. Schedule de la solución.

Habiendo analizando los resultados de la síntesis de la solución planteada, ya que cumplen con los requisitos se puede exportar el IP Core. Para ello se utiliza la opción Export RTL, especificando que se busca exportar un IP Core.

Luego queda importar el IP Core en Vivado para incluirlo en el diagrama de bloques. El mismo se incluye luego de la corrección gamma (con el AXI-Stream de salida de la corrección gamma como input). Es importante aclarar que la señal de control *ap_start* debe ser seteada a 1 para indicarle al IP Core que está listo para ejecutar. También se le deben conectar el clock y el reset del sistema. En la Figura 20 se muestra como se realiza la conexión.

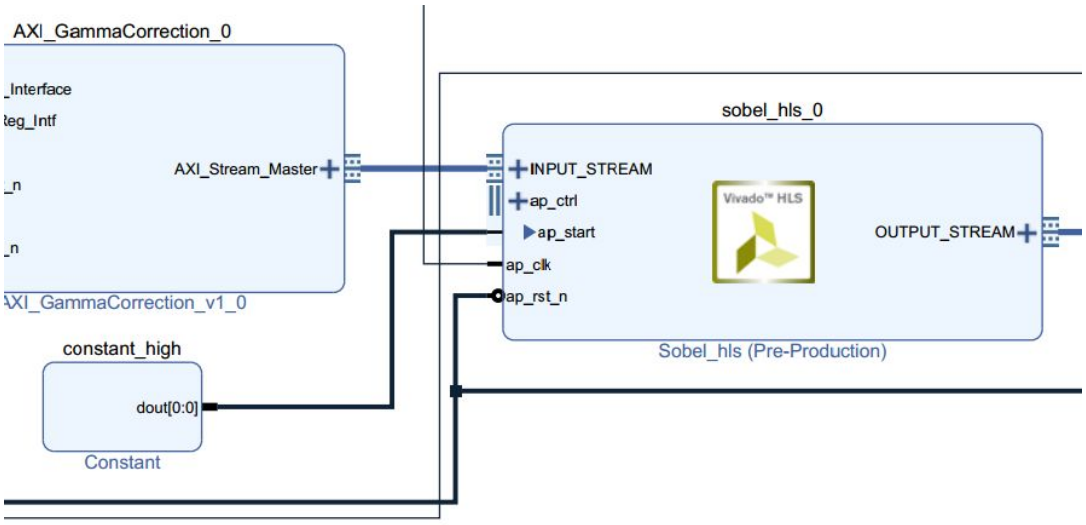

Fig. 20. IP Core exportado incluido en el diagrama de bloques.

Tras esto, queda re-sintetizar y re-implementar el diseño en Vivado, generar el bitstream, y seguir los mismos pasos ya descritos en Vivado SDK para tener el diseño funcionando en la placa.

5. Resultados

Tras haber seguido estos pasos, se obtuvieron las imágenes mostradas en la Figura 21 (*pass-through*) y en la Figura 22 (sobel). En ambos casos se pudo obtener videos de hasta 1080p a 60 fps. Se destaca la velocidad que tiene la plataforma para procesar el video, ya que es relativamente poco el tiempo entre la captura y la visualización en el monitor.

Fig. 21. Imagen obtenida del *pass-through*.

Fig. 22. Imagen obtenida del sobel.

Concluyendo, en este reporte se describen y analizan los pasos para poner una funcionamiento una plataforma de procesado de video. Se realizaron pruebas de un pass-through y un sobel con los que se obtuvieron buenos resultados.

Esta plataforma sienta las bases para poder rápidamente desarrollar y probar IP Cores desarrollados en HLS para ser corridos en la Zybo Z7-20 con la Pcam 5C. Esto será de vital importancia para otros pasos del proyecto de análisis de defectos en baldosas.