elektron ámbar

Original Article

Revista elektron, Vol. 4, No. 2, pp. 81-86 (2020)
https://doi.org/10.37537/rev.elektron.4.2.109.2020

Plataforma para Procesamiento de Imágenes sobre SoC FPGA de Xilinx

Platform for the Development of Image Processing Applications on Xilinx SoC FPGAs

Tomás Medina [#1], Lucas Leiva [#*2], Martín Vázquez [#*3]

[#] *LabSET - INTIA, Fac. Cs. Exactas, Universidad Nacional del Centro de la Prov. de Buenos Aires*
Pinto 399, (7000) Tandil, Pcia. De Buenos Aires, Argentina

[*] *Universidad Nacional de Tres de Febrero,*
Valentín Gómez 4828, (1678) Caseros, Pcia. De Buenos Aires, Argentina

[1] medinatomasariel@gmail.com
[2] lleiva@labset.exa.unicen.edu.ar
[3] mvazquez@labset.exa.unicen.edu.ar

Recibido: 14/10/20; Aceptado: 27/11/20

***Resumen*— La aplicación de sistemas de procesamiento de imágenes en *edge computing* resulta cada vez más atractiva y necesaria. Sin embargo, las exigencias en cuanto a consumo de potencia y alto rendimiento impiden que puedan utilizarse plataformas de procesamiento estándares. En este aspecto, los FPGA son una buena opción para el desarrollo de sistemas de visión computacional a causa de su capacidad de explotación del paralelismo. Por otra parte, el flujo de diseño de las herramientas de síntesis de FPGA actuales admiten lenguajes de alta abstracción como descripciones de entrada, en contraposición a los lenguajes de descripción de hardware. La síntesis de alto nivel (HLS) automatiza el proceso de diseño al transformar la descripción algorítmica en hardware digital mientras se satisfacen las limitaciones del diseño. Sin embargo, a los expertos en procesamiento de imágenes puede resultarles compleja la integración hardware obtenida con el resto de los componentes del sistema, como por ejemplo interfaces de captura y visualización. En este trabajo, se presenta un diseño base para la construcción de aplicaciones de procesamiento de imágenes basada en Zynq. Se proporciona además una metodología que posibilita el desarrollo eficiente de soluciones de procesamiento de imágenes embebidas de manera ágil.**

Palabras clave: procesamiento de imágenes; HLS; Zynq.

***Abstract*— The use of image processing systems is becoming frequent and is appropriate in edge computing. However, the demands on power consumption and high performance prevent the use of standard processing platforms. Thus, FPGAs are a good option for the development of computer vision systems because they are capable of exploiting parallelism. On the other hand, the design flow of current FPGA synthesis tools supports high-level languages as input descriptions, in contrast to hardware description languages. High-level synthesis (HLS) automates the design process by transforming an algorithmic description into digital hardware meeting design limitations. However, image processing experts may find challenging the hardware integration with the rest of the system components, e.g. capture and display interfaces. This work presents a basic design for the construction of image processing applications based on Zynq and a guide for its use, which solves this problem, allowing the agile generation of embedded image processing solutions. Additionally, a methodology is provided, which facilitates the efficient development of image processing embedded solutions in an agile way.**

Keywords: image processing, HLS, Zynq.

I. INTRODUCCIÓN

El procesamiento de imágenes y videos digitales es un área dinámica con aplicaciones críticas respecto al tiempo. Las mismas se encuentran cotidianamente en casos como drones militares y comerciales, sistemas de visión por computadora, vigilancia de áreas sensibles, control de acceso, inspección automatizada en industrias entre otras [1]-[3]. Si bien en estas aplicaciones la tasa de procesamiento es importante, existen otras restricciones como la capacidad de operación en tiempo real, el consumo de potencia, o el nivel de integración del sistema. La complejidad de procesamiento de estos algoritmos restrictivos hace que su implementación en procesadores de propósito general convencionales no sea factible y se adapten mejor a implementaciones en hardware específico [4].

La tecnología FPGA ha evolucionado significativamente en los últimos años, tanto en la escala de integración como en su extensión a SoC (*System-on-a-Chip*) FPGA, que incorporan arquitecturas de procesadores multinúcleo heterogéneas. Si bien el origen de la tecnología se enfocó en el uso exclusivo para aplicaciones electrónicas, actualmente son utilizados para el co-diseño hardware/software e implementación de sistemas complejos completos debido a las capacidades que proveen. Dentro de estas capacidades se destacan la disponibilidad para realizar miles de millones de operaciones MAC (acumulación de multiplicaciones) por segundo utilizando los cores DSP incluidos y los bloques de memoria BRAM, y las oportunidades para explotar los diferentes niveles de paralelismo que la mayoría de las aplicaciones de computación intensiva exigen [5].

Si bien el uso de esta tecnología de hardware resulta adecuado en aplicaciones de procesamiento de imágenes y video [6], cada vez es más complejo lograr un aprovechamiento de los recursos a medida que la escala y la

sofisticación de estos dispositivos aumenta, considerando el incremento de requerimientos de usuario (conectividad, precisión, resolución, interoperabilidad, entre otros). Tradicionalmente, lograr el rendimiento necesario ha requerido el diseño manual de circuitos personalizados a nivel de transferencia de registro (RTL) en un lenguaje de descripción de hardware. Este es un enfoque muy eficaz, pero impone una gran carga de desarrollo debido al bajo nivel de abstracción del diseño.

Con el fin de aumentar la productividad y promover las FPGA a una comunidad de usuarios más amplia, en los últimos años se han presentado nuevas metodologías de diseño con gran abstracción, incluida la síntesis de alto nivel (*High-Level Synthesis*) [7]. Esta metodología acepta el diseño en varios lenguajes (por ejemplo C, C++ y SystemC) y genera un nivel de transferencia de registro (RTL) sintetizable con precisión de ciclo a través de transformaciones de código y optimizaciones de síntesis. Dentro de las ventajas ofrecidas por esta tecnología de síntesis se destaca su capacidad de lograr una implementación del diseño a partir de una abstracción de alto nivel sin requerir un conocimiento exhaustivo del hardware, así como también la posibilidad de realizar una exploración del espacio de diseño con pocas modificaciones del algoritmo. Además las herramientas ofrecen métodos de depuración y verificación convenientes que disminuyen el tiempo de desarrollo. Estos métodos de entrada de diseño se han utilizado recientemente en una variedad de aplicaciones (por ejemplo, imágenes médicas, redes neuronales convolucionales y aprendizaje automático), con beneficios significativos en términos de rendimiento y consumo de energía [8][9]. En [10] se presenta un relevamiento de experiencias de diseño HLS vs. RTL demostrando que esta nueva metodología permite obtener más rendimiento y un uso de recursos FPGA ligeramente menor, con un incremento considerable en la productividad, a costa de la pérdida de calidad de resultados (*QoR*).

El enfoque convencional para desarrollar aplicaciones de visión embebidas basadas en FPGA parte de la implementación y prueba de los algoritmos usando software de propósito general. Esto se debe a que tanto esta primera implementación como su verificación resultan más fáciles de realizar en una plataforma de software que directamente en hardware [11]. Cuando se alcanza la funcionalidad deseada de un algoritmo se debe crear una descripción RTL del mismo, pudiendo ser este paso automatizado a través de herramientas de síntesis de alto nivel [12].. Existe una amplia gama de herramientas para este tipo de síntesis disponibles, que difieren significativamente tanto en su facilidad de uso como en la calidad del hardware derivado [13].

Si bien las metodologías de desarrollo utilizadas usualmente permiten a usuarios sin experiencia en hardware generar soluciones para sus algoritmos con un esfuerzo mínimo, ahorrando tiempo de desarrollo y reduciendo el riesgo de cometer errores [14], éstas no contemplan la integración de la implementación de algoritmos con los diferentes componentes hardware que integran el sistema. En este sentido, resulta propicio considerar la disponibilidad de una configuración base que permita a los expertos en aplicaciones de procesamiento de imágenes, integrar las soluciones algorítmicas de forma sencilla.

Existen varios trabajos en la literatura que abordan la construcción de plataformas de código abierto para el diseño de soluciones de procesamiento de imágenes basadas en FPGAs. Por ejemplo, en [15] se detalla una plataforma reconfigurable en Virtex-4SX35 para la detección de bordes a partir de los datos capturados de una cámara OV7610. En [16][17] se han propuesto otros proyectos de código abierto para el procesamiento de imágenes utilizando Altera DE2-115. En [18] se presenta otra solución que añade un entorno de diseño y validación, utilizando como entrada la información de imágenes capturada por un dispositivo OV7670, y presenta resultados utilizando una placa Nexys 4 FPGA.

Este trabajo propone una configuración inicial destinada a usuarios con poca experiencia en el diseño de sistemas en hardware que requieran implementar sistemas de procesamiento de imágenes sobre esta tecnología, y que pueda también ser utilizada como soporte para el desarrollo rápido de aplicaciones de procesamiento de imágenes. Además, presenta una guía para su uso, disponible en [19]. La propuesta se basa en el uso de la placa de desarrollo Zybo Z7-20 [20]. La plataforma admite la incorporación de algoritmos de procesamiento de imágenes desarrollados en C++, y mediante la herramienta de síntesis Vivado HLS, genera el bloque hardware correspondiente al algoritmo. Luego posibilita la incorporación de este de manera sencilla, en un diseño preestablecido que contiene la captura de imágenes, la visualización a través de una interfaz HDMI, y el soporte para el guardado de las imágenes de entrada en una tarjeta de memoria microSD. El objetivo principal del trabajo es brindar un entorno y una metodología que permitan reducir los inconvenientes iniciales respecto al tiempo de aprendizaje de la tecnología, y generar soluciones rápidamente.

En la sección II de este trabajo se presenta la descripción de la plataforma, explicando los componentes hardware, el diseño de la configuración y el soporte para el almacenamiento de imágenes en la memoria. La sección III describe la metodología para la implementación de algoritmos de procesamiento de imágenes. En la sección IV se presentan algunos de los resultados obtenidos, y finalmente en la sección V se detallan las conclusiones y trabajo futuro.

II. Diseño de la Plataforma

A. *Componentes Electrónicos*

La plataforma está basada en el uso de un sensor de captura de imagen Pcam 5C de Digilent [21]. El periférico cuenta con un sensor de imagen OV5640 con una montura de lente M12 que permite acoplar una óptica de acuerdo a las necesidades, e interfaz de conexión MIPI CSI-2 (*Mobile Industry Processor Interface Camera Serial Interface*).

Se seleccionó como unidad principal de procesamiento el kit de desarrollo Zybo Z7-20 que permite ser integrado con el sensor Pcam 5C. El equipamiento posee un SoC FPGA de Xilinx, Zynq XC7Z020-1CLG400C, interruptores de control, interfaces HDMI y microSD, que se consideran útiles para incluir funcionalidad en versiones futuras.

Figura 1. Componentes electrónicos seleccionados para la Plataforma.

La visualización de los resultados se realiza a través de una pantalla táctil LCD de 7 pulgadas de SpotPear Electronics [22], que posee interfaz HDMI y se acopla a la placa anteriormente mencionada. Los componentes de la plataforma se presentan en la Fig. 1.

B. *Diseño de Bloques*

El diseño de la configuración base fue realizado usando Xilinx Vivado 2018.2 utilizando la herramienta IP Integrator Block Design. La Fig. 2 representa el diagrama de bloques IP.

El pipeline de procesamiento de video incluye los siguientes conjuntos de módulos IP:

- *MIPI a AXI-Stream*: compuesto por los módulos IP *MIPI D-PHY* [23] y *MIPI CSI-2 Receiver* [24], se encarga de hacer la conversión de la señal proveniente del Pcam 5C a un AXI4-Stream. El formato de la conversión obtenida en esta etapa (crudo, obtenido del sensor o RGB) se puede configurar en tiempo de ejecución.
- *Bayer a RGB*: conformado por el módulo *IP Sensor Demosaic* [25], toma la información obtenida del sensor según el formato de Mosaico de Bayer [26] y la convierte a un espacio de color RGB. Esto es, en otras palabras, procesar los datos crudos que se obtienen del sensor de la cámara a un formato más conveniente para su posterior manejo (ya que comúnmente los módulos IP de procesado de imágenes y bibliotecas trabajan con imágenes RGB).
- *Corrección Gamma*: compuesto por el módulo IP Gamma Correction [27], realiza un ajuste para de la luminancia de la imagen y que los colores sean más perceptibles o diferenciables como en la visión humana. El factor de corrección puede ser configurado en tiempo de ejecución. De ser innecesaria para la aplicación, al establecer el factor de corrección en 1 la imagen no se verá modificada.
- *AXI VDMA*: conformado por el módulo *IP AXI Video Direct Memory Access* [28], se utiliza para comunicar la memoria principal con un AXI-Stream de video, con ancho de banda suficiente para alcanzar una resolución 1080p a 30 fps. Este componente es necesario para sincronizar los frames de videos salientes con la señal externa, para manejar cambios de frecuencia o resolución (ya que estos últimos se pueden configurar en tiempo de ejecución).

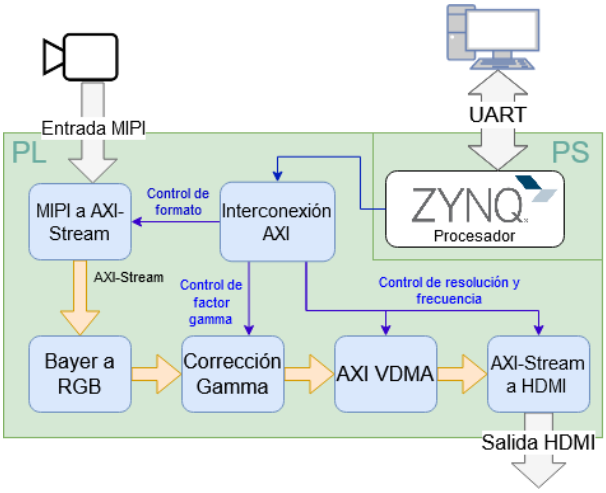

Figura 2. Representación simplificada del diagrama en bloques de la configuración Hardware.

- *AXI-Stream a HDMI*: compuesto por los módulos IP *AXI4-Stream to Video Out* [29] y *RGB to DVI Encoder* [30], convierte el AXI-Stream a una señal de video paralela (con tasa de cuadros configurable en tiempo real), y adapta esa señal para que sea transmisible por la salida HDMI.

Si bien la configuración del dispositivo de captura (resolución, tasa de cuadros) puede realizarse sobre la Lógica Programable (PL), se incluye al diseño el Sistema de Procesamiento Zynq (PS). Esto posibilita minimizar la complejidad en el desarrollo de aplicaciones de usuario, como en este caso permitiendo que las configuraciones se pueden establecer en tiempo real mediante UART, o en el futuro incorporar software que permita la transmisión de resultados sobre Ethernet. Además, el PS es el responsable de ejecutar las tareas de almacenamiento de imágenes brindado por la plataforma, descrito en la siguiente sección. El código fuente del PS fue desarrollado para ejecutar sobre *Bare Metal*.

Las configuraciones se establecen en los módulos IP mediante la comunicación por AXI-Lite entre el procesador y los demás módulos. Para que el PS se comunique con los múltiples módulos configurables de la PL, se incluye el módulo *IP AXI-Interconnect* [31] (Interconexión AXI en Fig. 2), que en este caso particular se utiliza para conectar el procesador en modo "Maestro" con los demás módulos en modo "Esclavo".

C. *Soporte para la Captura de Imágenes*

Si bien la plataforma provee facilidades para el desarrollo y testeo de algoritmos sintetizados como módulos IP descritos en lenguajes de alto nivel, esta propuesta considera la posibilidad de incorporar la captura de imágenes obtenidas del Pcam 5C en condiciones ambientales reales. Las imágenes pueden ser utilizadas como dataset en la etapa de simulación y co-simulación en Vivado HLS para testear los resultados parciales de algoritmos desarrollados y así hacer un ajuste más fino con resultados cercanos a los que se tendrá en la solución final embebida.

La captura de imágenes se realiza utilizando la interfaz microSD contenida en la placa de desarrollo, permitiendo operar con un único dispositivo de energía (como puede ser una batería portátil) sin necesidades de comunicación para

realizar las transferencias. La lógica de la funcionalidad para el soporte de captura de imágenes fue desarrollada en software ejecutado por el PS. El software almacena los frames almacenados por el AXI VDMA en la memoria principal utilizando las bibliotecas de Xilinx *XilFFS* (para acceder al sistema archivos FAT de la tarjeta de memoria), y *Xil_io* (para la lectura y escritura en memoria principal). Por otro lado, se utilizaron los pulsadores presentes en la placa para la toma controlada por el usuario. La configuración de esta funcionalidad utiliza el driver *XGPioPs*, que habilita una capa de software para la abstracción del manejo de GPIO conectados al PS.

El flujo de funcionamiento del programa consiste en la lectura de la pulsación del botón de la placa, que inicia la copia del frame de video en la memoria. Esto último es necesario ya que la velocidad de transferencia hacia la tarjeta microSD es menor a la del VDMA, por lo que interfiere con la transmisión por HDMI. Una vez duplicado el frame, se transfiere a la tarjeta microSD. El flujo de operación se presenta en la Fig. 3.

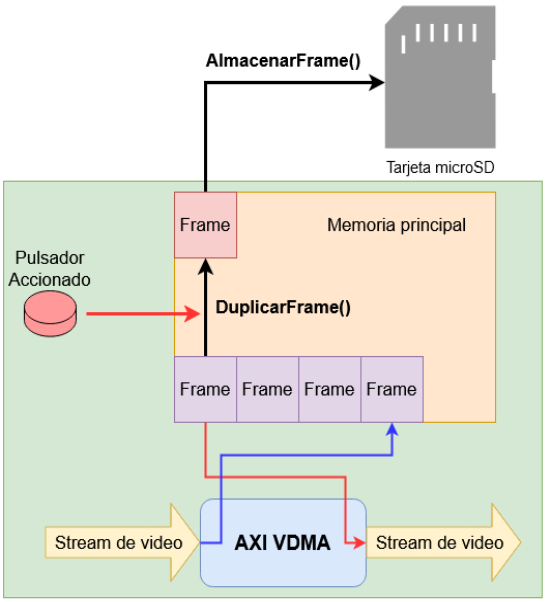

Figura 3. Proceso de almacenamiento del frame de video desde la memoria principal hacia la tarjeta microSD

III. Metodología para Desarrollo de Algoritmos de Procesamiento de Imágenes

El procedimiento para el diseño de aplicaciones de procesamiento embebidas se presenta en la Fig. 4, la cual permite mediante los recursos provistos generar ágilmente soluciones de acuerdo a las necesidades de la aplicación.

El diseño comienza a partir del uso de la biblioteca OpenCV [32]. La misma ofrece un amplio catálogo de funciones para sistemas de procesamiento de imágenes, y es particularmente familiar a los desarrolladores de aplicaciones de procesamiento de imágenes. De esta forma, el desarrollador puede comenzar a implementar una versión en alto nivel, y evaluar los resultados del algoritmo aplicado sobre un conjunto de imágenes de testeo obtenido desde la plataforma a través del uso del soporte para la captura de imágenes provisto. En este aspecto, se puede verificar en etapas tempranas la funcionalidad correcta del algoritmo en las condiciones ambientales reales (resolución, iluminación, ruido, etc).

Una vez alcanzado el nivel de precisión requerido, el algoritmo se migra a Vivado HLS, adaptando las funciones de OpenCV utilizadas a las provistas en la biblioteca *xfOpenCV* [33] de Xilinx. La misma incluye un conjunto de más de 60 kernels, optimizado para FPGA y SoC FPGA de Xilinx, basado en *OpenCV*. De esta forma, los algoritmos de procesamiento de imágenes pueden mantenerse descritos en un lenguaje como C o C++.

Para que el módulo IP generado pueda acoplarse en el diseño, es necesario definir las entradas y salidas como AXI-Streams, y usar las funciones provistas por *xfOpenCV* para procesar la imagen según se necesite. Además, se pueden codificar funciones propias que trabajen sobre las imágenes por el alto nivel de abstracción del lenguaje. A fin de lograr una mayor eficiencia en términos temporales se recomienda utilizar la directiva *Dataflow*, considerando sus limitaciones descritas en [34]. Esta directiva es compatible con las interfaces AXI-Stream y permite explotar la concurrencia de ejecución de las funciones, que se encuentran optimizadas dentro de la biblioteca. Se deben tener en cuenta durante la etapa de optimización ciertas características que inciden en el diseño del algoritmo, tales como la capacidad de operación concurrente del hardware, la longitud de los datos que puede ser variable y adaptadas a las necesidades, y la existencia de memoria distribuida que permite implementar un hardware apropiado, de acuerdo a los requerimientos y la disponibilidades de recursos físicos contenidos en el FPGA.

Una de las ventajas principales del entorno de desarrollo es su capacidad de simulación de los resultados por software. Luego de obtener los resultados deseados en C, se pueden utilizar las directivas provistas por el entorno para configurar la síntesis del módulo IP, realizando una exploración del diseño rápida en cuanto a rendimiento y consumo de recursos. Además, la herramienta permite la cosimulación, que compara los resultados de la simulación en software respecto a los del módulo IP sintetizado, pudiendo utilizar como conjunto de testeo las imágenes capturadas desde la plataforma.

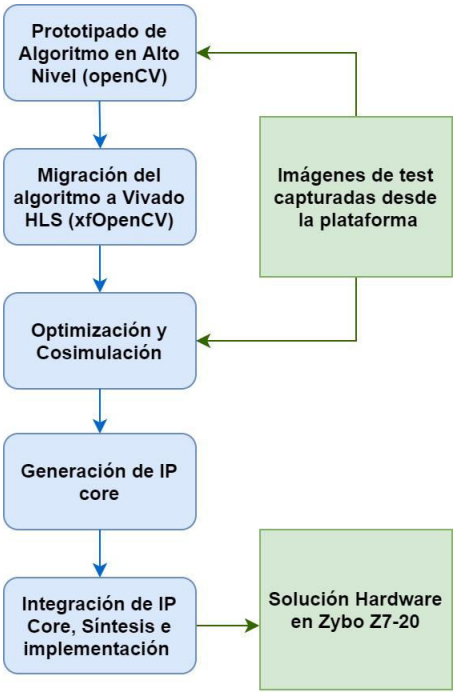

Figura 4. Metodología propuesta para el desarrollo de aplicaciones embebidas de procesamiento de imágenes usando Zybo Z7-20

Finalmente, el módulo IP es exportado e incluido en el pipeline de procesamiento de video del diagrama de bloques de Vivado (Fig. 2), ubicándolo preferentemente entre los bloques de *Corrección Gamma* y *AXI VDMA*. El diseño puede sintetizarse e implementarse para la configuración de la placa, obteniendo de esta manera una solución embebida para el procesamiento de imágenes.

IV. Resultados Experimentales

La primera prueba experimental fue un *Pass-through* de las imágenes obtenidas desde la cámara al puerto HDMI, sin incluir ningún módulo IP de procesamiento. Por otro lado, se evaluó con una aplicación de detección de bordes (Fig. 5 y Fig. 6), que incluye operaciones tales como *rgb2gray*, *sobel* y *addWeighted*. Además se implementó un algoritmo de detección de esquinas que aplica las funciones *rgb2gray*, *erosion*, *dilation*, *threshold* y *Harris-Stephens corner detection.*

En los tres casos, las pruebas se realizaron con una resolución de 1080p y 30 frames por segundo (fps). Los porcentajes de utilización de la lógica programable se muestran en la Tabla 1, donde se aprecia que existe un gran margen para la inclusión de más lógica de procesamiento. Se debe considerar que el *throughput* en todos los casos es igual a la tasa de cuadros de entrada (30 fps), y la latencia de la detección de bordes es de 13.8 ms (*throughput* máximo de 72 fps), mientras que para la detección de esquinas es de 27.9 ms (*throughput* máximo de 35 fps).

Figura 5. Imagen de entrada para la aplicación de detección de bordes.

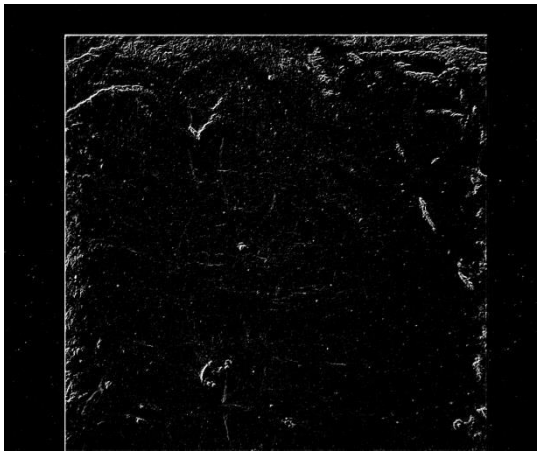

Figura 6. Imagen resultante de la aplicación de detección de bordes.

TABLA I
Porcentajes de Ocupación para los Casos de Estudio

	LUT	FF	BRAM	DSP
Pass-through	31%	22%	20%	0%
Detección de bordes	32%	22%	21%	1%
Detección de esquinas	43%	28%	45%	6%

V. Conclusiones y Trabajos Futuros

Este trabajo propone una plataforma, una guía de usuario y una metodología, para el desarrollo de sistemas de visión embebidos basada en Zybo Z7-20. La propuesta permite integrar de manera sencilla módulos IP generados a partir de descripciones de algoritmos complejos en alto nivel utilizando Vivado HLS. Se destacan las capacidades ofrecidas por este entorno tanto para la simulación como para la cosimulación. Por otra parte, el uso de una descripción algorítmica en lenguajes como C o C++ disminuye el tiempo de desarrollo, permitiendo a los expertos en algoritmos de procesamiento de imágenes de alto nivel una rápida adaptación a la tecnología.

Los resultados de síntesis demuestran que el uso de recursos es bajo (31% de ocupación de LUTs) considerando que la PL restante permite la incorporación de algoritmos para la implementación de sistemas más complejos, como el que implementa la detección de esquinas. La solución propuesta se implementó con éxito en dos aplicaciones: en

un sistema de detección de malezas en cultivos y en la detección de defectos en una línea de producción de baldosas cerámicas. En el futuro se espera aplicarla en un sistema de control parasitario en veterinaria ganadera. Se comprobó que la disponibilidad del soporte detallado en este trabajo permitió acortar el tiempo de desarrollo en las dos aplicaciones realizadas.

Se propone a futuro adaptar la plataforma para incluir la integración con otros dispositivos de captura como cámaras con interfaz USB, migración a otras placas de desarrollo e incorporar capacidades de transmisión de datos y resultados a través de la interfaz Ethernet.

AGRADECIMIENTOS

Este trabajo fue parcialmente financiado por la SeCAT de UNICEN (Código de Proyecto 03/C287) y la Secretaría de Investigación de la Universidad Nacional de Tres de Febrero.

REFERENCES

[1] J. Wang, Z. Feng, Z. Chen, S. A. George, M. Bala, P. Pillai, S.-W.Yang, and M. Satyanarayanan, "Edge-based live video analytics for drones," *IEEE Internet Computing,* vol. 23, no. 4, pp. 27–34, 2019.

[2] A. Heredia and G. Barros-Gavilanes, "Video processing inside embedded devices using ssd-mobilenet to count mobility actors," in *2019 IEEE Colombian Conference on Applications in Computational Intelligence (ColCACI).* IEEE, 2019, pp. 1–6

[3] H. Kavalionak, C. Gennaro, G. Amato, C. Vairo, C. Perciante,C. Meghini, and F. Falchi, "Distributed video surveillance using smart cameras,"*Journal of Grid Computing*, vol. 17, no. 1, pp. 59–77, 2019.

[4] J. Hosseinzadeh and M. Hosseinzadeh, "A comprehensive survey on evaluation of lightweight symmetric ciphers: hardware and software implementation," *Advances in Computer Science: an International Journal,* vol. 5, no. 4, pp. 31–41, 2016.

[5] F. Siddiqui, S. Amiri, U. I. Minhas, T. Deng, R. Woods, K. Rafferty,and D. Crookes, "Fpga-based processor acceleration for image processing applications,"*Journal of Imaging*, vol. 5, no. 1, p. 16, 2019.

[6] J. McAllister, "Fpga-based dsp," in *Handbook of Signal Processing System*s. Springer, 2010, pp. 363–392.

[7] P. Coussy and A. Morawiec,High-level synthesis: from algorithm to digital circuit. Springer Science & Business Media, 2008.

[8] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, 2015, pp. 161–170.

[9] D. Passaretti, J. M. Joseph, and T. Pionteck, "Survey on fpga in medical radiology applications: Challenges, architectures and programming models," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 279–282.

[10] S. Lahti, P. Sjövall, J. Vanne and T. D. Hämäläinen, "Are We There Yet? A Study on the State of High-Level Synthesis," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 38, no. 5, pp. 898-911, May 2019, doi: 10.1109/TCAD.2018.2834439.

[11] D. G. Bailey, *Design for embedded image processing on FPGAs*. John Wiley & Sons, 2011.

[12] D. G. Bailey, "The advantages and limitations of high level synthesis for FPGA based image processing," in *Proceedings of the 9th International Conference on Distributed Smart Cameras*, 2015, pp. 134–139.

[13] L. Huang, D.-L. Li, K.-P. Wang, T. Gao, and A. Tavares, "A survey on performance optimization of high-level synthesis tools," *Journal Of Computer Science and Technology*, vol. 35, pp. 697–720, 2020.

[14] M. Qasaimeh and E. N. Salahat, "Real-time image and video processing using high-level synthesis (hls)," in *Handbook of Research On Advanced Concepts in Real-Time Image and Video Processing*. IGI Global, 2018, pp. 390–408.

[15] M. K. Birla, "Fpga based reconfigurable platform for complex image processing," in *2006 IEEE International Conference on Electro/Information Technology*. IEEE, 2006, pp. 204–209.

[16] C. Ababei, S. Duerr, W. J. Ebel Jr, R. Marineau, and M. G.Moghaddam, "Open source digital camera on field programmable gate arrays," *International Journal of Handheld Computing Research(IJHCR)*, vol. 7, no. 4, pp. 30–40, 2016.

[17] C. Ababei, S.Duerr, W.J. Ebel Jr, R. Marineau, "Open source digital camera on field programmable gate arrays,"*International Journal of Handheld Computing Research(IJHCR)*, vol. 7, no. 4, pp. 30–40, 2016.

[18] X. Yang, Y. Zhang, and L. Wu, "A scalable image/video processing platform with open source design and verification environment," in *20th International Symposium on Quality Electronic Design (ISQED)*.IEEE, 2019, pp. 110–116

[19] Tomás Medina Github repository (2020), Zybo Z7-20 Video Processing Platform, [Online]. Available: https://github.com/Tomasmed18/Zybo-Z7-20-Video-Processing-Platform

[20] *Zybo Z7 Board Reference Manual,* Digilent, Zybo Z7-20, 2018.

[21] *Pcam 5C Reference Manual*, Digilent,2018

[22] (2020) SportPear Electronics, Raspberry Pi 7 inch HDMI LCD GPIO Touch (1024*600), [Online]. Available: http://www.spotpear.com/index/study/detail/id/165.html

[23] *MIPI D-PHY Receiver 1.3 IP Core User Guide,* Digilent, 2018.

[24] *MIPI CSI-2 Receiver 1.1 IP Core User Guide*, Digilent, 2018.

[25] *Sensor Demosaic v1.0 (Rev. 3),* Xilinx, 2018.

[26] Bryce E. Bayer. "*Color Imaging Array*". US Patent 3971065, 1976.

[27] *Gamma Correction v7.0 LogiCORE IP Product Guide PG004,* Xilinx, 2015.

[28] *AXI Video Direct Memory Access V6.3,* Xilinx, 2017.

[29] *AXI4-Stream to Video Out v4.0,* Xilinx, 2017.

[30] *RGB-to-DVI (Source) 1.4 IP Core User Guide,* Digilent, 2017.

[31] *AXI-Interconnect v2.1,* Xilinx, 2017.

[32] Team, OpenCV (2018) OpenCV library. [Online]. Available: https://opencv.org/about.html.

[33] Xilinx OpenCV User Guide UG1233, Xilinx, 2019.

[34] Vivado Design Suite User Guide: High-Level Synthesis, Xilinx, 2018.