ISBN 978-987-46297-7-7

Implementación de Filtro de Detección de Bordes Sobel en SoC usando Síntesis de Alto Nivel

Roberto Millon y Emmanuel Frati
Departamento de Ciencias Básicas y Tecnológicas, UNdeC
Chilecito (5360), La Rioja, Argentina
{rmillon,fefrati}@undec.edu.ar

Enzo Rucci
III-LIDI, Facultad de Informática, UNLP
La Plata (1900), Bs As, Argentina
erucci@lidi.info.unlp.edu.ar

***Resumen*—Las FPGAs se han destacado como dispositivos de bajo consumo energético y alta productividad para muchas aplicaciones de procesamiento de imágenes. En este trabajo se presenta la implementación de un filtro de detección de bordes Sobel usando síntesis de alto nivel sobre una plataforma SoC. A diferencia de otras propuestas, esta implementación no requiere de un sistema operativo embebido en la placa ni tampoco de librerías específicas de imágenes.**

***Index Terms*—Detección de bordes, Sobel, SoC, Zybo, HLS**

I. Introducción

El objetivo principal del procesamiento digital de imágenes es mejorar la calidad de una imagen o extraer información contenida en la misma [1]. En ese sentido, las FPGAs se han destacado como dispositivos de bajo consumo energético y alta productividad para muchas aplicaciones de procesamiento de imágenes [2].

Dentro del procesamiento de imágenes, las técnicas de detección de bordes o contornos son empleadas para segmentar una imagen en regiones homogéneas [3]. Estas técnicas se aplican sobre imágenes en escala de grises representadas por una función $f(x, y)$, donde f se corresponde con la intensidad o nivel de gris en cada píxel. Un borde está constituido por un cambio abrupto de intensidad en una región reducida y es lo que facilita delimitar los objetos presentes en una imagen.

El filtro (u operador) Sobel es un método de detección de bordes muy popular que ha sido ampliamente estudiado. En el ámbito de las FPGAs, las implementaciones del filtro Sobel han sido desarrolladas tanto en lenguajes de descripción de hardware (HDL) como en lenguajes de alto nivel (HLL). En HDL, se pueden destacar tres diseños de este operador. En [4] se desarrolló una versión simple del filtro en VHDL y se realizaron pruebas de rendimiento en 3 FPGAs de Xilinx de distinta gama, mientras que en [5] se desarrolló una versión multidireccional de este operador. Las pruebas de rendimiento de este último se realizaron en una FPGA Spartan3 XC3S200. En cuanto a Verilog, se encuentra un diseño del filtro implementado en un FPGA Virtex 4 de Xilinx [6]. Por otro lado, vale la pena mencionar 2 versiones HLS del filtro Sobel. En primer lugar, en [7] se implementa el algoritmo en una plataforma System-on-Chip (SoC) de Avnet-Xilinx denominada Zedboard. Las imágenes ingresan por una cámara USB al sistema y los resultados se envían al puerto HDMI para ser visualizados en pantalla. El procesador es el responsable de controlar todos los componentes del sistema, para lo cual cuenta con un sistema operativo (SO) Linux embebido. En segundo lugar, en [8] se realizan pruebas de la implementación en otra plataforma SoC denominada ZYBO, del consorcio Digilent-Xilinx. En este caso, se utilizan tanto los puertos HDMI como VGA para el ingreso de las imágenes y su posterior procesamiento. Una característica en común de ambos desarrollos de alto nivel es que los autores utilizaron funciones de librerías de video provistas por Xilinx.

En este trabajo se presenta la implementación de un filtro Sobel sintetizado en lenguaje de alto nivel sobre una plataforma SoC. A diferencia de las propuestas existentes, la implementación presentada no requiere embeber un SO en la placa ni tampoco utiliza funciones de librerías de imágenes/video. Además, se encuentra disponible en un repositorio web público [9] para beneficio de la comunidad, lo que facilita su reuso o modificación para otros fines.

II. Filtro de Detección de Bordes Sobel

El filtro Sobel es un método de detección de contornos basado en gradientes (o de primer orden) que se aplica a imágenes en escala de grises [10]. Utiliza dos máscaras de convolución independientes M_h y M_v para obtener los cambios de intensidad en dirección horizontal y vertical, respectivamente, que se pueden observar en la Eq. 1.

$$M_h = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} M_v = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad (1)$$

Desde el punto de vista matemático, el operador emplea las máscaras para aplicar convolución a la imagen original y así calcular aproximaciones a las derivadas tanto en el eje horizontal como en el eje vertical. En cada píxel de la imagen, los resultados de las aproximaciones de ambos gradientes pueden ser combinados para obtener su magnitud (o tasa de cambio de intensidad) con la siguiente ecuación $G = \sqrt{{G_x}^2 + {G_y}^2}$.

III. Implementación

Para la implementación del filtro Sobel se emplearon dos clases de arreglos diferentes denominados *buffer de línea* y *ventana deslizante*, siguiendo las recomendaciones de [11].

ISBN 978-987-46297-7-7

Por un lado, los buffers de línea almacenan una fila completa de píxeles de la imagen y se usa la cantidad mínima que permita mantener el contexto necesario para procesar cada píxel. Por otro lado, la ventana deslizante almacena un conjunto de píxeles centrado en el píxel de interés de forma de poder aplicar la convolución correspondiente. En este caso particular, se requiere una ventana deslizante de 3×3 y 3 buffers de línea de *AnchoDeImagen* elementos, como se muestra en la Fig. 1.

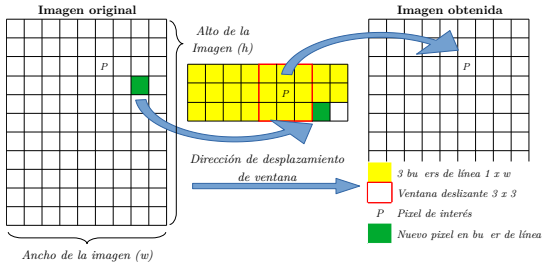


Figura 1: Aplicación de convolución

El núcleo de procesamiento está formado por 3 bloques sintetizados y validados en alto nivel con la herramienta Vivado HLS v2019.1 de Xilinx. El primer bloque (RGB2GRAY) convierte imágenes RGB en escala de grises utilizando el método basado en la media aritmética de los tres componentes; el segundo bloque (FILTRO SOBEL) detecta los bordes en la imagen; y el tercer bloque (U8toU32) combina 4 caracteres de 8 bits para formar una palabra de 32 bits. Se utilizaron directivas para la optimización de la síntesis. En particular, se usó `HLS INTERFACE` para indicar que se desea emplear los estándares AXI Stream y AXI Lite en las comunicaciones de las imágenes y sus parámetros. También se empleó `HLS ARRAY PARTITION` para permitir la lectura de las memorias en paralelo (buffers de línea, ventana deslizante y máscaras). Por último, se incluyó la directiva `HLS UNROLL` para que las convoluciones puedan computarse en paralelo.

Para realizar pruebas reales del filtro se diseñó un sistema que permite adquirir imágenes RGB en formato BMP desde una tarjeta de memoria microSD, detectar sus bordes y almacenar la nueva imagen en la memoria microSD. El sistema está formado por un bloque DMA que accede a la memoria microSD para la lectura de la imagen y envía el flujo de píxeles al bloque de procesamiento. Luego de operar, el bloque de procesamiento devuelve el flujo de píxeles al DMA para su posterior almacenamiento en memoria. La configuración y habilitación del sistema son realizadas por el procesador. En la Fig. 2 se observa un diagrama en bloques del sistema. Por último, todos los módulos fueron integrados usando la herramienta Vivado 2019.1.

IV. Resultados Experimentales

La plataforma de desarrollo usada es ZYBO, basada en la arquitectura AP SoC e integrada por un procesador *dual-core* ARM Cortex-A9 y una FPGA XC7Z010-1-CLG400C. La aplicación de prueba se desarrolló con la herramienta XSDK 2019.1 de Xilinx y se configuró la implementación para que

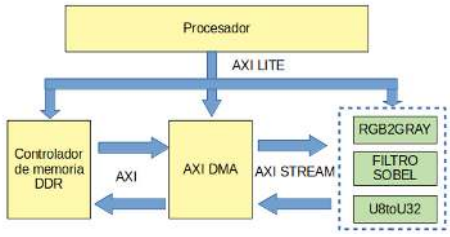
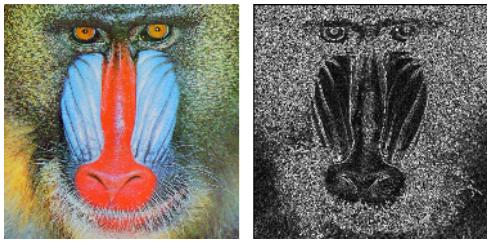


Figura 2: Sistema implementado



(a)

(b)

Figura 3: Imágenes de prueba. (a) *Mandrill*. (b) *Kodim23*

pueda procesar imágenes de hasta 1920×1080 píxeles (Full HD). El sistema fue probado con 2 imágenes obtenidas de repositorios públicos: *Mandrill* de 512×512 [12] y *Kodim23* de 768×512 [13]. La Fig. 3 muestra el resultado del filtrado aplicado las imágenes de prueba.

El análisis temporal reporta que el tiempo de ciclo es 8.62*ns* mientras que el retardo es de 3.96*ms*. El tiempo requerido para procesar *Kodim23* a 100Mhz es de 40.97*ms*. Con respecto al uso de recursos, la Tabla I resume los valores requeridos. Como se puede notar, el diseño no es grande y quedan aun recursos disponibles para otras aplicaciones si fuese necesario.

Recurso	Slice LUTs	Slice Registers	F7 Muxes	BRAM	DSPs
Total	17600	35200	8800	60	80
Usados	2877	3221	6	1	2
Utilización	16 %	9 %	1 %	2 %	3 %

Tabla I: Uso de recursos

V. Conclusiones y Trabajo Futuro

En este trabajo se presentó la implementación de un filtro Sobel sintetizado en lenguaje de alto nivel sobre una plataforma SoC, la cual no requiere de un SO embebido en la placa ni tampoco librerías específicas de imágenes. Además, se encuentra disponible en un repositorio web público para beneficio de la comunidad. Como trabajo futuro, se espera desarrollar una versión del mismo algoritmo en HDL y realizar un análisis comparativo de las mismas considerando uso de recursos y rendimiento alcanzado.

REFERENCIAS

[1] M. Pavan Kumar, "Hardware Acceleration of Edge Detection Using HLS," 2019. [Online]. Available: http://scholarworks.csun.edu/bitstream/handle/10211.3/210950/Murthy-Pavan%20Kumar-thesis-2019.pdf?sequence=1

[2] L. Daoud, D. Zydek, and H. Selvaraj, "A Survey of High Level Synthesis Languages, Tools, and Compilers for Reconfigurable High Performance Computing," in *Advances in Systems Science*, J. Swiatek, A. Grzech, P. Swiatek, and J. M. Tomczak, Eds. Cham: Springer International Publishing, 2014, vol. 240, pp. 483–492. [Online]. Available: http://link.springer.com/10.1007/978-3-319-01857-7_47

[3] T. Acharya and A. K. Ray, *Image Processing: Principles and Applications*. John Wiley & Sons.

[4] G. Chaple and R. D. Daruwala, "Design of Sobel operator based image edge detection algorithm on FPGA," in *2014 International Conference on Communication and Signal Processing*. Melmaruvathur, India: IEEE, Apr. 2014, pp. 788–792. [Online]. Available: http://ieeexplore.ieee.org/document/6949951/

[5] A. Nosrat and Y. S. Kavian, "Hardware description of multi-directional fast sobel edge detection processor by VHDL for implementing on FPGA," vol. 47, no. 25, pp. 1–7. [Online]. Available: http://research.ijcaonline.org/volume47/number25/pxc3879872.pdf

[6] V. Sanduja and R. Patial, "Sobel edge detection using parallel architecture based on FPGA," vol. 3, p. 5.

[7] Y. Zheng, "The design of sobel edge extraction system on FPGA," vol. 11, p. 08001. [Online]. Available: http://www.itm-conferences.org/10.1051/itmconf/20171108001

[8] A. Ben Amara, E. Pissaloux, and M. Atri, "Sobel edge detection system design and integration on an FPGA based HD video streaming architecture," in *2016 11th International Design Test Symposium (IDT)*, pp. 160–164, ISSN: 2162-061X.

[9] "Implementación de filtro de detección de bordes sobel en soc usando hls." [Online]. Available: https://github.com/robertoamt/Sobel_HLS_ZYBO

[10] N. Nausheen, A. Seal, P. Khanna, and S. Halder, "A FPGA based implementation of Sobel edge detection," *Microprocessors and Microsystems*, vol. 56, pp. 84–91, Feb. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0141933116302289

[11] F. M. Vallina, "Implementing memory structures for video processing in the vivado HLS tool," p. 8.

[12] "The usc-sipi image database." [Online]. Available: http://sipi.usc.edu/database/database.php

[13] "Kodak image dataset." [Online]. Available: http://www.cs.albany.edu/~xypan/research/snr/Kodak.html